Conceção e desenvolvimento de interfaces gráficas baseadas em sistema de microcontroladores Microchip

Rui Miguel Oliveira Ferreira Gomes

Dissertação do MIEM

Orientadores:
Prof. Manuel Rodrigues Quintas
Prof. Paulo Augusto Ferreira de Abreu

Mestrado Integrado em Engenharia Mecânica

Opção de Automação

Outubro 2013

Resumo

A disponibilidade no mercado de microcontroladores de baixo custo e de ecrãs tácteis, em conjunto com ferramentas de *software* livres, constitui uma oportunidade para o desenvolvimento de sistemas de controlo com interfaces gráficas.

Neste trabalho é realizado um protótipo, baseado em microcontroladores, de um sistema modular com ecrã táctil, dotado de um conjunto de menus que permitem o controlo da velocidade e posição angulares de um motor elétrico de corrente contínua (CC) com escovas, e a sua monitorização a partir do sinal de saída do *encoder* associado.

Foram realizadas a configuração do *software* e *hardware* de desenvolvimento da Microchip e, com base na sua biblioteca gráfica foram implementadas funcionalidades típicas em interfaces gráficas que possibilitam a interação com o módulo desenvolvido para o controlo do motor CC.

O presente relatório de dissertação inclui a apresentação das diversas tecnologias envolvidas no projeto, as arquiteturas de *hardware* de aplicações gráficas e também a biblioteca gráfica. É descrita a arquitetura base do projeto e respetiva implementação e posteriormente "liga-se" a HMI (*Human Machine Interface*) para apresentar os vários ecrãs desenvolvidos.

Finalmente são apresentadas conclusões e sugestões para trabalhos futuros.

Graphical interfaces design and development based on Microchip’s microcontrollers

Abstract

The market availability of low cost microcontrollers and displays, gathered with free software tools, represent an opportunity to develop control systems with graphical interfaces.

This project’s implements a microcontroller based prototype with touch screen menus for controlling a brushed DC motor´s angular velocity and position, and the actuator monitorization from an encoder signal.

Microchip microcontroller´s hardware and software were configured and based on its graphic library, the design and edition of typical functions in graphical interfaces allowing the implementation with the developed actuator’s control module.

The present dissertation report includes an introduction to the technologies involved in the project, the graphical applications hardware architecture and the graphic library. It also describes the project base architecture and implementation, and afterwards the HMI (Human Machine Interface) is “turned on” to explore the various developed menus screens.

Finally, the presentation of conclusions and future works suggestions.

Agradecimentos

Gostaria de agradecer aos meus orientadores Prof. Doutor Manuel Quintas e Prof. Doutor Paulo Abreu.

Queria também expressar o meu agradecimento ao Mestre Tiago Andrade.

Ao coordenador da opção Automação do MIEM, Prof. Doutor Francisco Freitas agradeço, e felicito as ações realizadas no âmbito da unidade curricular Dissertação.

Ao laboratório de acolhimento na secção de Automação, Instrumentação e Controlo do DEMec da FEUP, agradeço os recursos disponibilizados.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

ADC	*Analog to Digital Converter*
API	*Application Programming Interface*
bpp	*bits per pixel*
CC	Corrente Contínua
CNC	*Computer Numerical Control*
CS	*Chip Select*
DCS	*Distributed Control System*
DDD	*Display Device Driver Layer*
DEMec	Departamento de Engenharia Mecânica
DFLTCON	*Digital Filter Control*
DMA	*Direct Memory Access*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
ESK	*Ethernet Starter Kit*
FEUP	Faculdade de Engenharia da Universidade do Porto
GDC	*Graphic Display Controller*
GOL	*Graphic Object Layer*
GPIO	*General Purpose Input Output*
GPL	*Graphic Primitive Layer*
GPS	*Global Positioning System*
GPU	*Graphic Processor Unit*
GUI	*Graphical User Interface*
HMI	*Human Machine Interface*
LCD	*Liquid-Crystal Display*
MCU	*Microcontroller Unit*
MEB	*Multimedia Expansion Board*
MFM	*Motion Feedback Module*

MIEM	Mestrado Integrado em Engenharia Mecânica
MIPS	*Million Instructions per Second*
OSCCON	*Oscillator Control register*
PADCFG1	*Pad Configuration Control register*
PCB	*Printed Circuit Board*
PDIP	*Plastic Dual In-line Package*
PIM	*Plug-in Module*
PMADDR	*Parallel Master Port Address register*
PMAEN	*Parallel Master Port Enable register*
PMCON	*Parallel Master Port Control register*
PMMODE	*Parallel Master Port Mode register*
PMP	*Parallel Master Port*
PMSTAT	*Parallel Master Port Status register*
POSCNT	*Position Counter register*
PPS	*Peripheral Pin Select*
PWM	*Pulse Width Modulation*
QEI	*Quadrature Encoder Interface*
QEICON	*Quadrature Encoder Interface Control register*
QVGA	*Quarter Video Graphics Array*
RAM	*Random Access Memory*
RGB	*Red Green Blue*
SCADA	*Supervisory Control and Data Acquisition*
SPI	*Serial Peripheral Interface*
SSP	*Synchronous Serial Port*
STN	*Super-twisted Nematic*
TFT	*Thin-Film-Transistor*
TTL	*Transistor-Transistor Logic*
VGA	*Video Graphics Array*
WVGA	*Wide Video Graphics Array*

1 Introdução e objetivos

1.1 Interfaces gráficas

Uma HMI permite disponibilizar informação, alertas e comandos para um utilizador interagir com uma máquina ou sistema de automação. O *hardware* utilizado pela HMI é normalmente equipamento dedicado, embora possam ser utilizados dispositivos de utilização quotidiana, como um *tablet*.

A HMI permite a interpretação da informação transferida do sistema de um modo intuitivo e *user-friendly*, sendo o comando e monitorização do sistema tão eficaz quanto o projeto e implementação da solução de interface. Em acréscimo, as HMI são passíveis de inclusão no projeto original da máquina ou sistemas de automação, ou mesmo posteriormente em projetos de remodelação tecnológica motivada pela necessidade de dotar sistemas de produção de maior capacidade e eficiência ou implementar normas de segurança e higiene no trabalho e/ou ambientais.

As HMI podem encontrar-se no domínio industrial como elemento de interface de sistemas SCADA (*Supervisory Control and Data Acquisition*) e DCS (*Distributed Control System*) que controlam e monitorizam processos (e.g., Figura 1.1), ou em máquinas singulares (Figura 1.2), designadamente CNC (*Computer Numerical Control*).

Figura 1.1 – Soluções HMI industrial da *Beijer electronics*

Figura 1.2 – HMI industrial

Na aviação militar e comercial as interfaces Humano/Máquina, constituem um elemento importante na prevenção de acidentes. O comandante e a sua equipa de pilotagem devem dispor de forma intuitiva de informação de posição, velocidade e aceleração, da rota da aeronave, de elementos de monitorização das turbinas, além de representação de rotas por GPS, comunicações com outras aeronaves e aeroportos. Esta tecnologia com elevada complexidade é incluída em toda a aeronáutica [1], inclusive nas aeronaves de menor dimensão (e.g., Figura 1.3).

Figura 1.3 – HMI *Garmin* G1000® em aeronave de pequena dimensão

As marinhas de defesa nacional e mercante também dispõem nos seus navios de tecnologia HMI concebida em materiais com capacidade de resistência à corrosão. Em embarcações de menor dimensão, sem ponte de pilotagem, como na náutica de pesca desportiva e turismo as HMI implementadas têm ecrã antirreflexo, como as representadas na Figura 1.4.

Figura 1.4 – HMI para navios da marinha mercante

O controlo e monitorização de motores elétricos para, por exemplo, o acionamento de ventiladores nos automóveis ou de um eixo de acionamento em aplicações fabris constituem outras áreas onde as HMI são enquadráveis.

São múltiplos os domínios onde se podem encontrar as interfaces gráficas, sendo pertinente classificar e nivelar estes equipamentos em três grandes áreas de acordo com a dimensão do seu ecrã: elevada (12'' a 21''), média (7'' a 12'') e baixa (menor que 7'') [2].

É possível criar e desenvolver sistemas específicos combinando componentes avulso e integrando-os através de *hardware* e *software*. A Figura 1.5 apresenta um exemplo de uma aplicação que através da associação de um servomotor, um *drive*, uma HMI, e de uma aplicação de *software* (e.g., *Windows embedded* CE), disponíveis no mercado, implementa uma solução industrial de acionamento elétrico.

Figura 1.5 – Acionamento elétrico com HMI da *Parkermotion*

Outro exemplo de conceção de HMI para comando e monitorização de atuadores é a integração de um ou vários microcontroladores e um *display* gráfico táctil, através de *hardware* e *software* que inclua bibliotecas gráficas.

1.2 Microcontroladores

Segundo a *Smithsonian Institution*, o primeiro microcontrolador foi implementado em 1971 pelos engenheiros Gary Boone e Michael Cochran, na *Texas Instruments*. Uma etapa importante no percurso evolutivo dos MCU (*Microcontroller Unit*) foi a introdução das EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) em 1993, *e.g.*, o PIC16x84 da Microchip.

A evolução dos MCU conduziu a uma diminuição da sua dimensão e do seu custo, ao aumento da eficiência energética e, consequentemente, à sua implantação crescente em múltiplos domínios da atividade humana.

Os microcontroladores (Figura 1.6) estão dotados de valências que permitem a interação com vários tipos de periféricos. Uma área de aplicação importante para microcontroladores é o comando de atuadores, designadamente motores [3]. A tecnologia, de *hardware* e *software*, necessária ao microprocessamento de informação para uma interface gráfica com o utilizador de um motor, também pode ser baseada em microcontroladores.

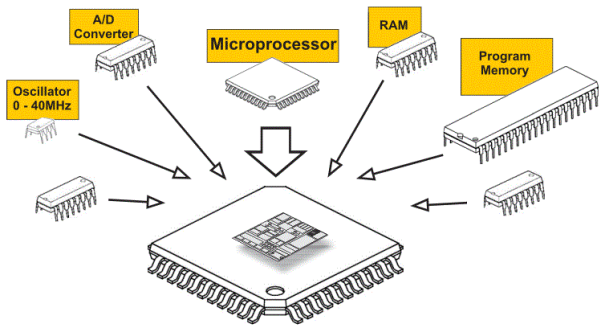


Figura 1.6 – Esquema dos periféricos integrados num microcontrolador

1.3 Tecnologias de ecrãs tácteis

O ecrã táctil, ou *touch screen* na Língua de William Shakespeare, é um tipo de ecrã sensível à pressão de um dedo ou de uma caneta, sendo muito utilizado como periférico de entrada de dados, em alternativa ao uso de um teclado ou rato.

É difícil imaginar que há algumas décadas atrás a tecnologia *touch screen*, estava somente presente em livros e filmes/séries de ficção científica, *e.g.*, *Star Trek* que constitui um exemplo do modo como se antecipava o futuro na interação entre Humanos e Máquinas.

Atualmente, esta tecnologia é utilizada em muitos domínios, designadamente, nas indústrias automóvel, aeronáutica, na domótica e nos equipamentos do dia-a-dia tais como telemóveis, computadores, *tablet*, etc. Foram necessárias várias gerações e evoluções tecnológicas (Figura 1.7) para que o ecrã táctil atingisse o elevado grau de implantação que detém na atualidade [4].

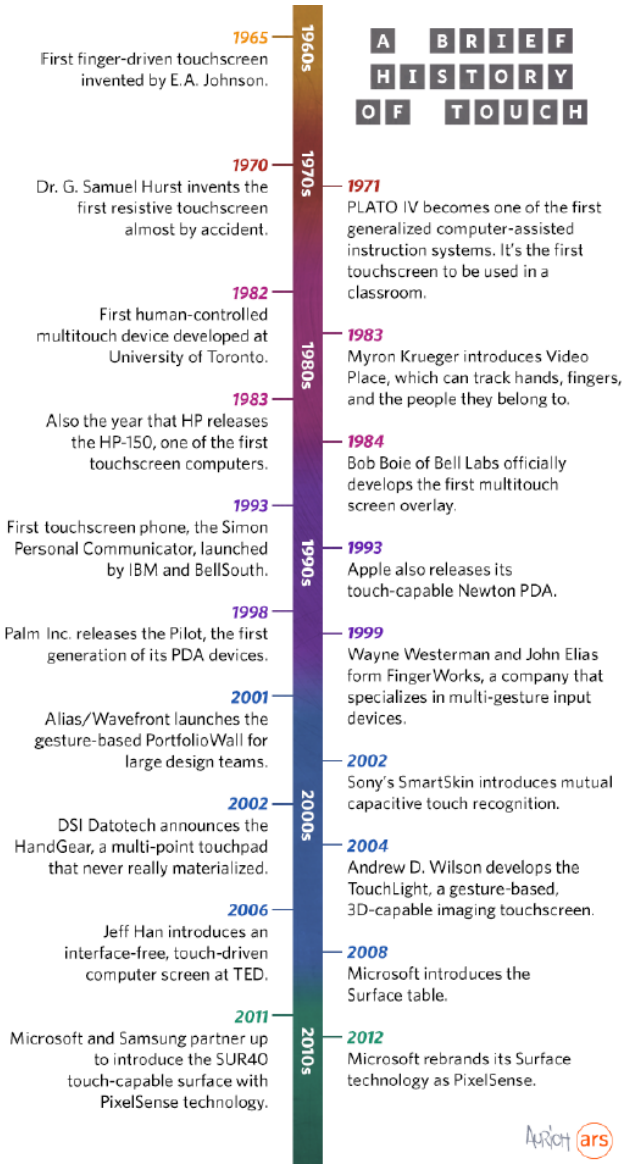


Figura 1.7 – Evolução cronológica do ecrã táctil

O primeiro ecrã táctil, Figura 1.8, foi inventado em 1965 por E. A. Johnson, no *Royal Radar Establishment* – Malvern, Reino Unido [1].

Figura 1.8 – Primeiro ecrã táctil

Na década de 1970, com o propósito de estudar Física Atómica sem recorrer a um acelerador *Van de Graaff*, foi concebida uma aplicação em compósito de papel condutor para leitura de pares de coordenadas XY pelo Dr. G. Samuel Hurst [4]. Este investigador, em conjunto com a sua equipa no *Oak Ridge National Laboratory* – Tennessee (EUA), perante a evidência de que um ecrã táctil no computador constituía uma excelente forma de interação utilizador e máquina desenvolveram uma nova interface. Para esse efeito colocaram uma película condutora de cobertura que, quando pressionada sobre a placa que alberga o eixo cartesiano 2D, induz uma queda de tensão entre os elementos condutores X e Y que permitia o cálculo das coordenadas, *i.e.*, da posição. Assim se iniciou a evolução que conduziu à tecnologia táctil resistiva.

O *hardware* associado a um *touch screen* inclui: um sensor, um circuito integrado e o *display*. O sensor é o elemento principal do dispositivo pois permite a identificação e comunicação ao circuito integrado, da posição do dedo ou caneta. Existem vários tipos de sensores, sendo os do tipo resistivo os mais utilizados devido ao seu baixo custo, boa resolução de imagem, e à imunidade elevada relativamente a agentes externos, designadamente, água, luz e poeiras [5].

Há técnicas para implementação da tecnologia *touch* resistiva com 8, 7, 6, 5 ou 4 condutores. Segundo a empresa *Elo Touch Solutions* a solução com 5 condutores apresenta maior índice de durabilidade, sendo a técnica“4-*wire*”, conforme Figura 1.9, a mais simples de implementar. Esta consiste na ligação do ADC (*Analog to Digital Converter*) às GPIO (*General Purpose Input Output*) do

microcontrolador da aplicação. Quando as soluções são mais exigentes, designadamente no processamento de dados, inclui-se um componente específico [6], como, por exemplo, o AR1000 da Microchip ou o MAX11803 da *Maxim*.

Para ativar o *touch*, a pressão de contacto do dedo ou caneta diminui a camada delgada de ar entre o filme de polímero e a camada subjacente, induzindo uma variação da tensão. A posição é calculada com base nessa variação de tensão.

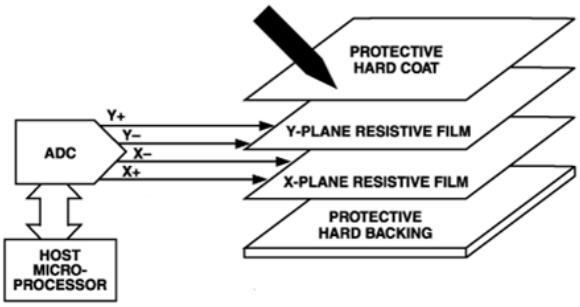


Figura 1.9 – Multicamadas de filme resistivo e revestimentos de proteção do ecrã táctil

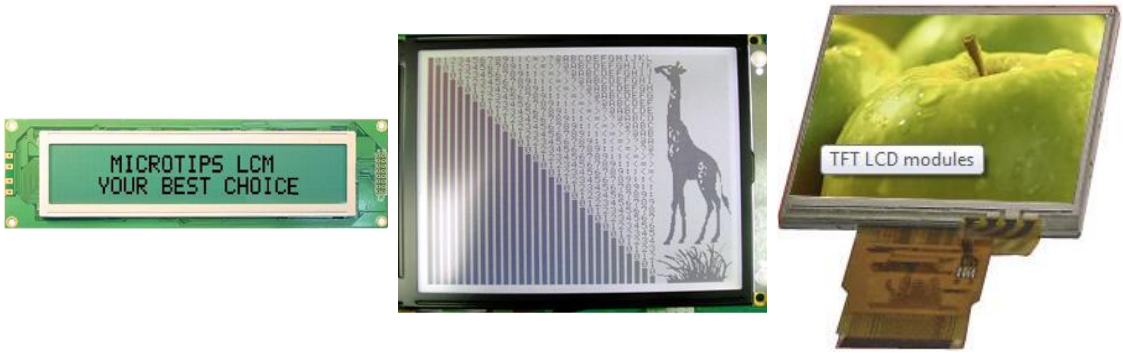
A colocação do *touch screen* sobre o *display* é um processo que ocorre em ambientes com qualidade do ar controlada pois não podem ficar retidas partículas de pó no filme de ar interior do *touch*. Outro aspeto a ter em consideração é garantir que as cotas entre o *touch* e o *display* do LCD (*Liquid-Crystal Display*) estão no intervalo compreendido entre 0.2 e 0.5 mm [7]. Desta forma a empresa *NewhavenDisplay International* garante que não vai ocorrer a formação de anéis de Newton.

Figura 1.10 – Interface entre *touch* e *display*

Existem diversos tipos de ecrãs para aplicações gráficas, sendo muito utilizada a tecnologia LCD TFT (*Thin-Film-Transistor*), embora a opção STN (*Super-twisted Nematic*) represente um menor custo. As resoluções para ecrãs com as dimensões correspondentes às gamas baixa e média de HMI (referidas no subcapítulo 1.1) podem ser: VGA (*Video Graphics Array*), WVGA (*Wide Video Gra-*

phics Array) ou QVGA (*Quarter Video Graphics Array*), que corresponde a 320x240 *pixels*.

Os LCD de carateres, ecrã gráfico monocromático e ecrã gráfico policromático (Figura 1.11 **a**, **b** e **c** respetivamente) constituem a gama de produtos disponível na Microtips *Technology*. Para o tipo de ecrã gráfico policromático existe uma versão que inclui a tecnologia táctil resistiva.



(a) **(b)** **(c)**

Figura 1.11 – Os 3 tipos de LCD da *Microtips*: (a) carateres, (b) gráficos monocromático e (c) policromático

A Microchip disponibiliza *software* de bibliotecas gráficas para a Microtips *Technology*. As duas empresas referidas são representadas na península ibérica pela *Sagitron* (*sagitron.com/pt*), com sede em Madrid, que promove seminários técnicos gratuitos no Porto.

1.4 Motivação e objetivos

Uma das facetas mais evidentes da evolução tecnológica a que temos assistido, assenta na procura, cada vez mais intensa, de automatizar os processos de interface com utilizadores de Máquinas em vários domínios da atividade humana, por forma a simplificá-los.

Esta demanda crescente implica projeto e desenvolvimento mais expedito e menos oneroso de HMI fiáveis, seguras e robustas.

Os trabalhos baseados em microcontroladores anteriormente desenvolvidos, por discentes do plano de estudos regular ou de programas de intercâmbio universitários internacionais, na secção de Automação, Instrumentação e Controlo do Departamento de Engenharia Mecânica, têm incluído, no protótipo final, HMI do

tipo LCD de carateres, baseadas em microprocessadores, ou implementadas no computador.

Os resultados bastante satisfatórios alcançados nos protótipos referidos com periféricos: PWM (*Pulse Width Modulation*), MFM (*Motion Feedback Module*) ou SPI (*Serial Peripheral Interface*), constituem um excelente indicador e estímulo ao desenvolvimento com outros periféricos de microcontroladores que melhorem as interfaces com os utilizadores.

O objetivo principal do presente trabalho é a conceção e desenvolvimento de uma interface gráfica, baseada em microcontroladores, que possibilite o controlo de velocidade e posição angulares de um atuador elétrico, a partir de ecrã táctil.

O desenvolvimento da HMI deve retirar partido das vantagens da aplicação da tecnologia de microcontroladores, assim como da biblioteca gráfica, para a realização dos elementos pictóricos.

A arquitetura de *hardware* implementada deverá ser modular, incluindo um ou mais microcontroladores em cada módulo que comunicam entre si, podendo ocorrer *upgrades* de módulos para incluir, designadamente, sensores, existindo disponibilidade de tempo e/ou recursos.

Os ecrãs criados para a interface gráfica deverão interagir com o módulo de atuação desenvolvido laboratorialmente, que será constituído por uma parte física de eletrónica de sinal (para comunicação) e de potência para o motor, e uma componente de código de programação de *software*.

De referir que atingir estes objetivos, significa também, abordar a variação da velocidade, de um atuador elétrico, e a sua monitorização sem recurso a potenciómetro ou botoneiras e LCD de carateres.

1.5 Estrutura da dissertação

O relatório de dissertação está organizado em cinco capítulos. O primeiro capítulo apresentou uma introdução às tecnologias base do projeto: HMI, microcontroladores e ecrãs tácteis. O capítulo dois refere as arquiteturas e a comunicação no interior das interfaces gráficas, e o *software* de bibliotecas gráficas para Microchip. O terceiro capítulo apresenta o protótipo modular e a comunicação entre módulos, e inclui o *hardware* e *software* utilizados no seu desenvolvimento,

com maior ênfase no módulo desenvolvido de controlo de motor CC com escovas. A apresentação da interface gráfica, com o utilizador do tipo táctil, é efetuada no capítulo seguinte: o quarto. O quinto e último capítulo é dedicado às conclusões e sugestões para trabalhos futuros.

2 Interfaces gráficas baseadas em microcontroladores Microchip

2.1 Introdução

Após uma referência geral às interfaces gráficas no que concerne a características e áreas de aplicação, é necessário apresentar os elementos de *hardware* e *software* para o desenvolvimento do projeto da interface com o utilizador, assim como os tipos de comunicação de dados no interior das arquiteturas existentes.

A Microchip disponibiliza soluções que permitem a implementação de interfaces que incluem um ecrã gráfico com *touch* resistivo. Esta tecnologia foi a utilizada por ser a menos onerosa, quando comparada com as tecnologias capacitiva ou indutiva, incluídas na gama *mTouch* ^{TM} da Microchip.

O presente capítulo descreve os componentes de *hardware* base necessários para uma aplicação gráfica e as soluções de *software* de desenvolvimento gráfico da Microchip disponíveis no mercado.

2.2 Arquiteturas

O microcontrolador é um dos quatro componentes base para uma aplicação gráfica, sendo os restantes três: o *buffer* de imagens, o microprocessador gráfico e o *display* (Figura 2.1) [8], podendo, como se irá indicar mais adiante no presente subcapítulo, o processamento gráfico, e o respetivo microprocessador, ser substituído por "processamento gráfico direto do microcontrolador".

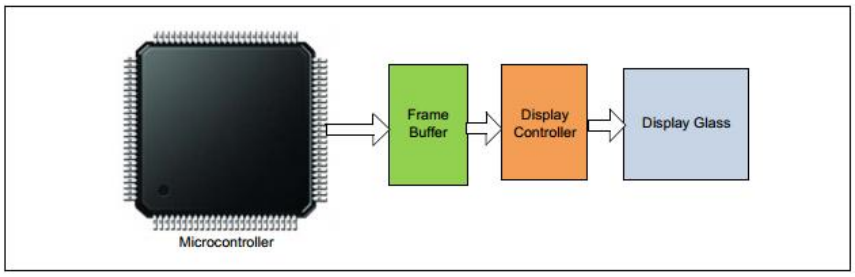


Figura 2.1 – Componentes que asseguram as funções base para implementação do *display* gráfico

O microcontrolador é o elemento principal da aplicação, e controla todos os outros elementos da arquitetura. É responsável, em conjunto com a biblioteca gráfica pela renderização dos objetos. O *frame buffer* é um componente para armazenagem de imagens. Assim, como o artista pode imaginar diversas cenas em muito menos tempo do que necessitaria para pintá-las, os microcontroladores também necessitam deste espaço de memória dedicado para armazenar imagens a fim de acelerar todo o processo de visualização no ecrã.

A dimensão do *frame buffer* é dada pela Equação 2.1.

$$size_{frame\ buffer} = \frac{n.º\ pixels\ \times color\ depth}{8} \times \frac{1}{1024} \quad [kbyte] \qquad \text{Equação 2.1}$$

A profundidade de cor (*color depth*) corresponde ao número de *bits* utilizados para representar a cor de um *pixel*.

O microprocessador gráfico é o responsável pela atualização das imagens no ecrã.

Existem três formas de integração dos componentes referidos para uma aplicação gráfica e disponibilizados no mercado pela Microchip, que serão abordadas até ao final do presente subcapítulo, designadas por "microprocessador gráfico externo", "unidade de processamento gráfico embebida" e "processamento gráfico direto".

A solução com microcontrolador e microprocessador gráfico externo, representada na Figura 2.2, é a que integra um maior número de componentes discretos: três, carece de mais espaço no PCB (*Printed Circuit Board*) e o custo do microprocessador externo poderá tornar a aplicação dispendiosa. Existe uma variante, em que o microprocessador gráfico externo está embebido no módulo de *display*

mas é a mais onerosa e na eventualidade de ser necessário a substituição do módulo de *display* e a sua produção tiver terminado, terá que ser desenvolvida toda uma nova aplicação.

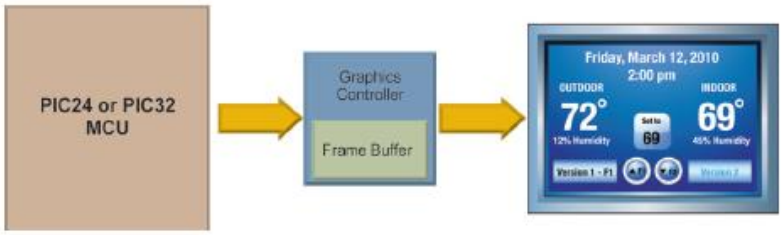


Figura 2.2 – Arquitetura com microcontrolador e microprocessador gráfico externo

Estas opções são exequíveis com PIC24, dsPIC e PIC32 e no que concerne ao microprocessador gráfico existem vários fabricantes, designadamente a Epson ou a *Solomon Systech*. Esta última empresa foi criada no final do século passado em Hong Kong e atualmente para além das infraestruturas no local onde foi fundada, dispõe de mais dois Centros tecnológicos na República Popular da China (Shenzhen e Beijing) e em Singapura.

A empresa *Solomon Systech* dispõe de *drives* de *display* genéricos, microprocessadores gráficos e, no topo da hierarquia, microprocessadores de imagem, cujas valências permitem implementar uma arquitetura do tipo da representada na Figura 2.3.

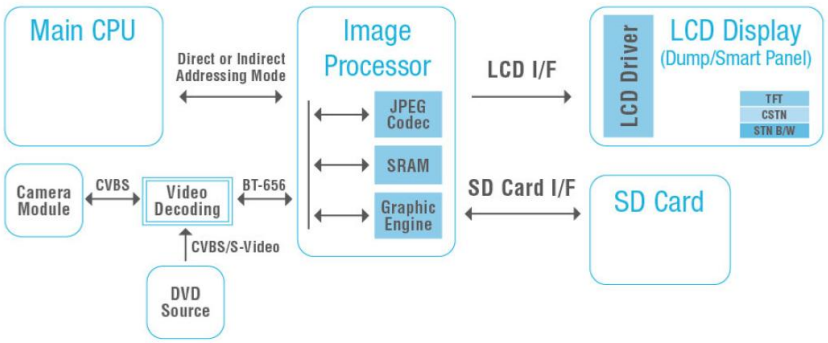


Figura 2.3 – Arquitetura com microcontrolador, microprocessador de imagem SSD1926 da *Solomon Systech* e LCD

Na configuração com unidade de processamento gráfico embebida no microcontrolador (Figura 2.4) a dimensão do *display* é limitada pela capacidade do *buffer* do microcontrolador dedicado às imagens.

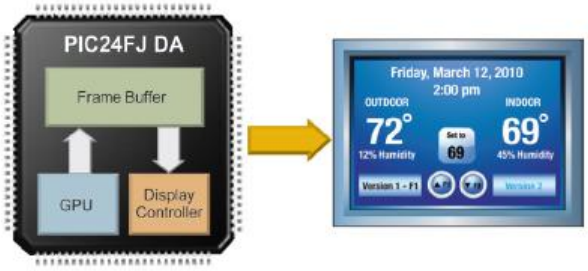


Figura 2.4 – Arquitetura com microcontrolador com unidade de processamento gráfico embebida

No entanto, a capacidade do *buffer* de imagens pode ser aumentada recorrendo à implementação de um *chip* extra externo de memória, resultando numa arquitetura com três dispositivos discretos (Figura 2.5).

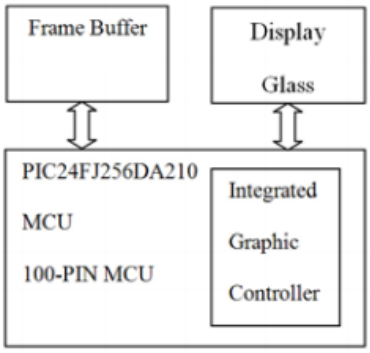


Figura 2.5 – Arquitetura com microcontrolador PIC24FJ256DA210 e *chip* extra externo

As arquiteturas referidas estão disponíveis exclusivamente para a família de MCU PIC24FJxxxDAxxx [9]. As características dos MCU desta família, para além das gráficas, devem ser estudadas pois podem não existir e/ou estar condicionadas para determinados periféricos, importantes para uma outra solução que se pretenda desenvolver e integrar na solução gráfica.

De acordo com o manual da *Sagitron* (de 2013) há uma terceira opção: "processamento gráfico direto" (Figura 2.6), para microcontroladores de 32 *bits*, o PIC32.

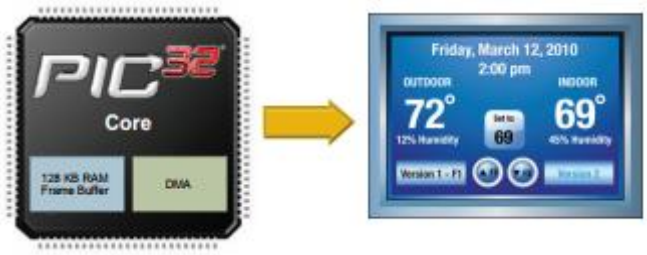


Figura 2.6 – Arquitetura com processamento gráfico direto

Esta gama de microcontroladores da Microchip disponibiliza 80 MIPS (*Million Instructions per Second*) e DMA (*Direct Memory Access*) de elevada performance para a renderização de objetos diretamente para os ecrãs. Assim, o módulo de *display* e o microcontrolador não são mediados pelo processador gráfico, comunicando diretamente.

O PIC32 é compatível com ecrã QVGA 8 bpp com *buffer* de imagens interno ou WQVGA 16 bpp com *buffer* de imagens externo. Dispõe de 512 kB e 128 kB RAM que permitem a inserção do código para outras aplicações além das gráficas e comunicação de dados, designadamente USB, CAN e *Ethernet*, em conjunto com valências gráficas. É possível implementar uma solução com *buffer* externo (Figura 2.7).

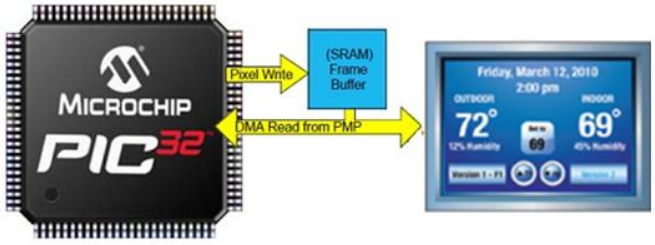


Figura 2.7 – Arquitetura com microcontrolador PIC32 e *chip* extra externo

Todas as características referidas tornam bastante apelativo o desenvolvimento com PIC32, em alternativa aos PIC24, em particular ao dedicado às aplicações gráficas (PIC24FJxxxDAxxx), referido no subcapítulo 2.2.

2.3 Comunicação

O tipo de arquitetura implementada e a comunicação de dados influenciam-se reciprocamente.

A comunicação em paralelo (Figura 2.8) caracteriza-se pelo envio em simultâneo de vários *bits* [10].

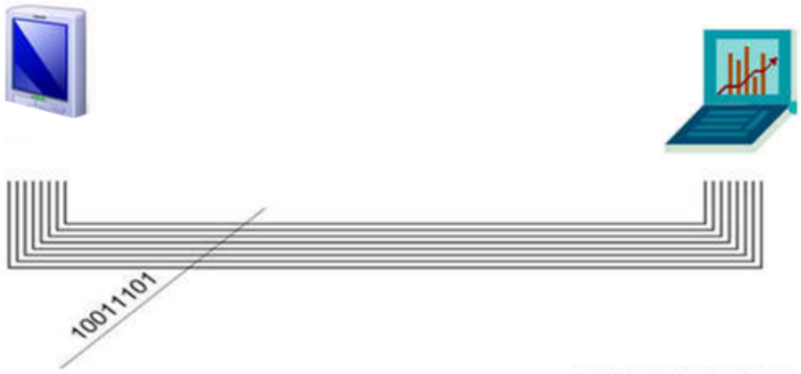


Figura 2.8 – Comunicação em paralelo

Este tipo de comunicação é utilizado em aplicações com processamento gráfico externo ao microcontrolador (Figura 2.9), para comunicação entre PIC32 ou PIC24 e o microprocessador gráfico.

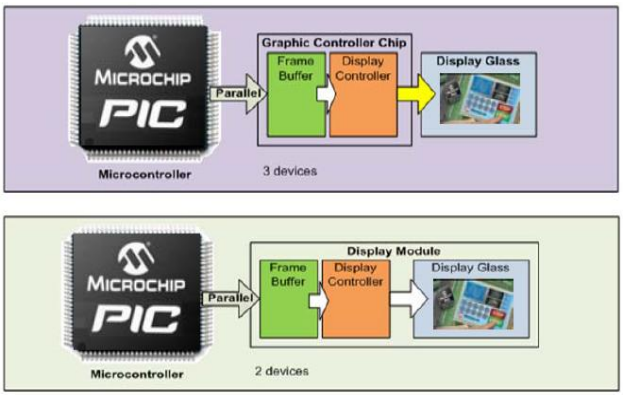


Figura 2.9 – Comunicação em paralelo para arquiteturas com microprocessador gráfico externo

Para interfaces gráficas desenvolvidas a partir de placas de desenvolvimento dedicadas da Microchip com arquiteturas de processamento gráfico externo, a implementação, como é óbvio, já está efetuada, contudo é necessário a verificação dos componentes efetivamente alocados a estas funções pois podem existir ligações não documentadas nas notas técnicas que inviabilizem a comunicação com outras aplicações exteriores à arquitetura de comando do *display*. Exemplo

disso é a placa MEB (*Multimedia Expansion Board*) JIT 111713075 em conjunto com PIC32 ESK (*Ethernet Starter Kit*) E320803 conforme testes efetuados e descritos no capítulo seguinte.

A Microchip desenvolveu para a comunicação com LCD o módulo PMP (*Parallel Master Port*) de entrada/saída, disponível em 8 *bits* no PIC24 (Figura 2.10) [11] e em 8 ou 16 bits no PIC32.

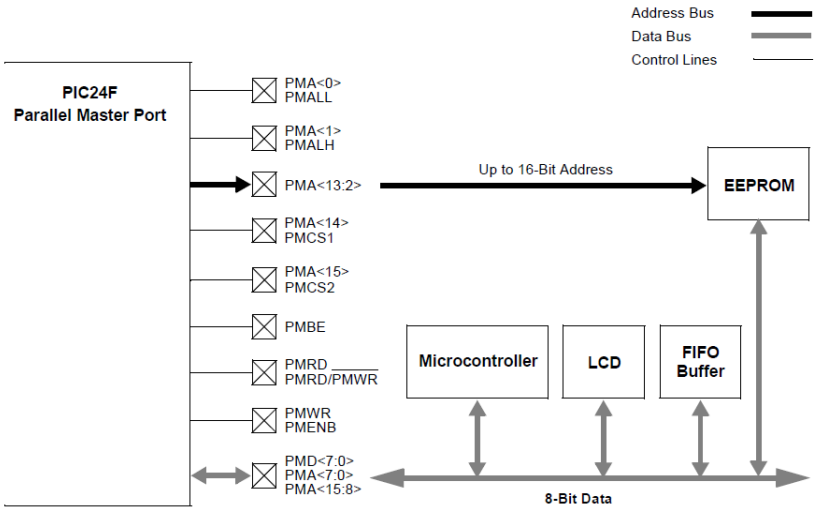


Figura 2.10 – Módulo PMP

Os registos associados são o PMCON (*Parallel Master Port Control register*) que contém os *bits* que controlam as funcionalidades básicas do módulo. O *bit* ON permite efetuar *reset* e habilitar ou desabilitar o módulo. Quando o estado corresponde a desabilitado todos os pinos I/O associados retornam à sua função original. Qualquer operação de leitura ou escrita ativa ou pendente é interrompida e o *bit* BUSY toma o valor zero. A informação no interior dos registos é retida, inclusive a contida no PMSTAT (*Parallel Master Port Status register*). Assim, é possível desativar o módulo após a receção de dados, sem perda da informação e do estado. O PMSTAT contém os bits de estado quando em modo *slave*. O PMMODE (*Parallel Master Port Mode register*) alberga os *bits* que controlam os modos operacionais: *master*/*slave*, assim como as suas opções de configuração. Inclui a *flag* de estado BUSY, utilizada no modo *master* para indicar que este modo está em utilização. O PMADDR (*Parallel Master Port Address register*) está encarregue das operações com endereços e CS (*Chip Select*). O

PMAEN (*Parallel Master Port Enable register*), através da ativação dos *bits* correspondentes, habilita os pinos do PMP. O *bit* 0 do registo PADCFG1 (*Pad Configuration Control register*): PADCFG1bits.PMPTTL, é um registo não específico do PMP mas afeta a sua configuração e permite a seleção entre TTL (*Transistor-Transistor Logic*) e ST (*Schmitt Trigger*) para *buffer* de entradas digitais.

Nas arquiteturas com processamento gráfico interno ao microcontrolador, este comunica diretamente com LCD TFT *touch* por RGB (*Red Green Blue*), conforme o esquema representado na Figura 2.11.

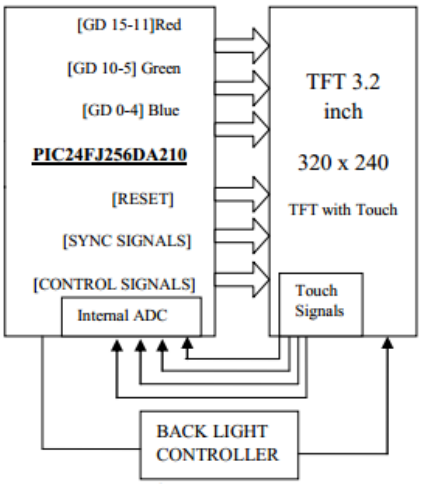


Figura 2.11 – Interface por RGB

Para a arquitetura com processamento gráfico direto, o barramento utilizado é também do tipo RGB.

2.4 Bibliotecas gráficas

A Microchip disponibiliza gratuitamente em *microchip.com* a sua biblioteca gráfica, com capacidade de gerar objetos 2D e 3D. Admite a utilização de periféricos do tipo *touch screen* e teclado.

A biblioteca gráfica está estruturada de uma forma modular (Figura 2.12), com uma base de *hardware* GDC (*Graphic Display Controller*), relacionada com a arquitetura implementada.

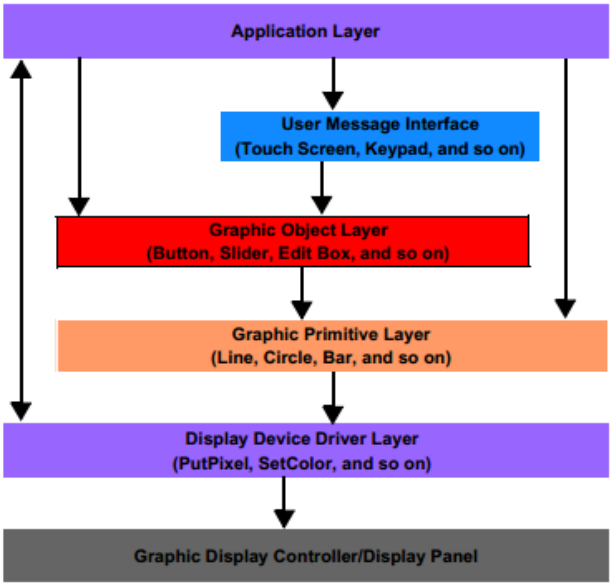


Figura 2.12 – A biblioteca gráfica Microchip

O nível da biblioteca imediatamente acima da GDC corresponde a DDD (*Display Device Driver Layer*), conforme a Figura 2.12. Todos os *displays* dispõem de um conjunto de comandos e indicadores de estado próprios. Assim, para cada tipo de *display* é necessário um *software* específico, que cumpra os requisitos de *hardware* do *display* e as API (*Application Programing Interface*) definidas pela biblioteca gráfica da Microchip. As funções principais disponibilizadas pelo nível DDD são por exemplo:

- *ResetDevice*() : função que permite a inicialização do sistema
- *SetColor*(color) e *PutPixel*(x,y) : função que altera a cor de um *pixel*
- *GetPixel*(x,y) : função que devolve a cor de um *pixel*

As outras funcionalidades disponibilizadas incluem a obtenção dos valores máximos para as abcissas e ordenadas. É possível implementar uma aplicação utilizando somente este nível. Nestes casos, todas as formas gráficas têm de ser definidas pelo programador. Para incluir no projeto este nível são necessários os seguintes ficheiros:

- *Graphics*.h
- *DisplayDriver*.h ou <Driver.h> específico, por exemplo, SSD1926.h que corresponde ao microprocessador gráfico *Solomon Systech* SSD1926

- *HardwareProfile*.h
- *GraphicsConfig*.h
- <Driver.c> específico, *e.g.*, SSD1926.c

O nível GPL (*Graphic Primitive Layer*) utiliza recursos através de API que permitem desenhar formas básicas. São exemplos destas formas, as linhas (contínua, larga, tracejada), as barras, o retângulo, o círculo, o polígono, o arco e concordância. A implementação deste nível carece da referência aos ficheiros: *Primitive*.h e *Primitive*.c.

A GOL (*Graphic Object Layer*) consiste em vários objetos selecionáveis designados por *widgets*. Estes podem ser do tipo, botão, caixa de texto, *check Box*, *scroll Bar*, barra de progresso, imagem, *listBox*, caixa de grupo, contador analógico, contador digital, gráfico e grelha, que constituem normalmente os elementos pictórios base de uma aplicação gráfica mais complexa.

A fonte e as cores das várias partes e/ou estados de cada objeto têm um estilo associado por defeito. Por exemplo, o botão, tem uma cor diferente conforme está ou não ativo. A GOL depende dos níveis DDD e GPL. A função encarregue da representação de objetos *GOLDraw*(), deve ser chamada num ciclo contínuo para simplificar o desenho dos *widgets*. A *GOLDrawCallback*() deve ser implementada no código da aplicação. Se a GOL for utilizada, no ficheiro *GraphicsConfig.h*, é necessário definir a macro #define USE_GOL.

A interface com o utilizador (*User Message Interface*) permite a transmissão de mensagens entre os *widgets* e os dados de entrada. É um subnível da GOL estando habilitado se a GOL estiver ativa. Por exemplo, se o utilizador pressionar um botão, a mensagem é enviada através da função *GOLMsgCallback*() que contém uma rotina de verificação de estado do botão “PRESSED” e indicação de uma ação.

No nível de aplicação (*Application Layer*) o utilizador tem o controlo pleno da aplicação. Deve ser efetuada a inicialização através da função *GOLInit*(), no caso de todos níveis estarem ativos, ou *InitGraph*(), se só estiverem a ser utilizados os níveis GPL e Interface com utilizador.

Outras bibliotecas gráficas disponibilizadas por outros fabricantes tais como, a *MICRIμM*® e a SEGGER *Microcontroller* também são compatíveis com micro-controladores Microchip.

A *MICRIμM*® foi fundada em Setembro de 1999. A sede desta empresa situa-se nos Estados Unidos da América, mais propriamente em Weston Florida e tem uma filial no Canadá, em Verdun – Quebec.

A biblioteca μC/GUI permite a criação de interfaces gráficas para aplicações embebidas. É possível implementar formas gráficas variadas, em diferentes tipos de tecnologias, desde um ecrã monocromático até ao *smartphone*. A utilização de ecrã táctil possibilita a elaboração de interfaces com o utilizador que tornam a aplicação mais apelativa.

Pode ser efetuado o *download* gratuito de uma versão de teste em *micrium.com* que carece do *software Microsoft Visual* C++ 6.0 ou superior para a elaboração de projetos. Na Figura 2.13 é possível visualizar uma aplicação para controlo de um veículo elétrico.

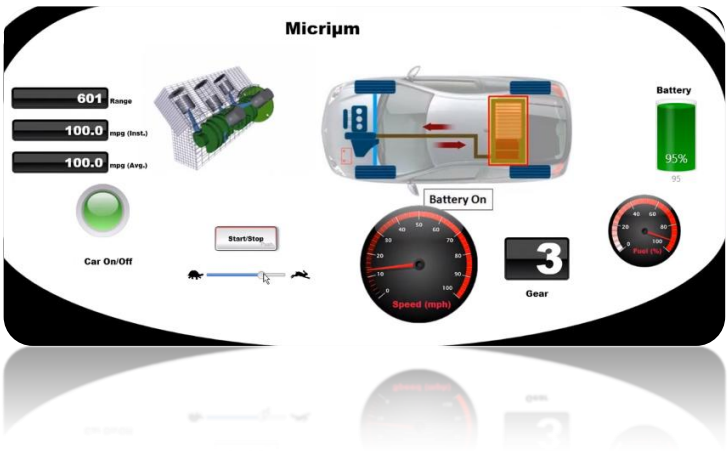


Figura 2.13 – Aplicação para controlo de veículo elétrico baseada na biblioteca gráfica da Micriμm

A SEGGER *Microcontroller* é uma empresa com sede na Alemanha em Hilden, e uma filial no estado de Massachusetts (E.U.A.) em Winchendon. Foi fundada em 1997 e dedica-se ao desenvolvimento e distribuição de *hardware* e respetivo código para ferramentas de desenvolvimento, e *software* para componentes de sistemas embebidos. O objetivo principal desta empresa é potenciar a celeridade no desenvolvimento de aplicações com microcontroladores, através da

disponibilização de ferramentas de fácil utilização e alta qualidade e custo reduzido. A aplicação informática *emWin* foi projetada para operar em ambientes dedicados ou generalistas, e tem a estrutura de *software* conforme a representada na Figura 2.14.

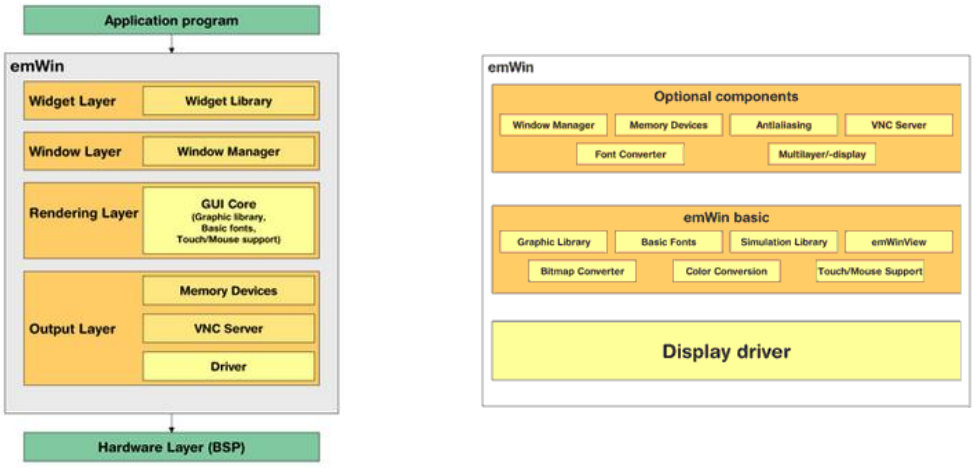


Figura 2.14 – Estrutura de *software* da biblioteca gráfica *emWin*

2.5 Conclusão

O presente capítulo fez referência às configurações de *hardware* possíveis para implementação de interfaces gráficas.

Apresentaram-se as arquiteturas de *hardware*, para PIC32 e PIC24, e *software* gráfico de desenvolvimento da Microchip, para aplicações com interface por *touch screen*.

A arquitetura com processamento gráfico externo apresenta-se como uma boa solução em termos de definição de imagens gráficas mas em engenharia os custos são um aspeto importante, podendo as arquiteturas com processamento gráfico embebido ou direto, apresentarem-se como melhores no compromisso entre qualidade e preço.

Foi efetuada uma introdução às bibliotecas gráficas que será complementada em capítulos subsequentes, permitindo que a assimilação do tema que é constituído por aspetos de organização e implementação de elementos virtuais, associados a *software* extenso, vá sendo intercalada com o desenvolvimento do *hardware* para a HMI e o controlo de motores CC, domínios claramente físicos.

3 Protótipo desenvolvido

3.1 Introdução

Este capítulo apresenta o protótipo desenvolvido, referindo a sua arquitetura baseada em microcontroladores, e de seguida são enumerados todos os componentes de *hardware* utilizados, assim como o *software* de desenvolvimento geral: que inclui as principais configurações do PIC18LF4431, e específico para aplicações gráficas, o GDDx.

Contém ainda os circuitos eletrónicos (de sinal e de potência) realizados, a implementação da comunicação SPI, o módulo PWM e submódulo QEI (*Quadrature Encoder Interface*), e o controlo de posição e velocidade angulares do motor CC com escovas.

3.2 Arquitetura e *hardware*

A aplicação implementada no projeto é constituída por uma placa de desenvolvimento Microchip que integra: o ecrã táctil, o microprocessador gráfico e o microcontrolador I, e, uma placa desenvolvida com o microcontrolador II que implementa o comando do atuador elétrico (Figura 3.1). A comunicação entre microcontroladores é do tipo SPI.

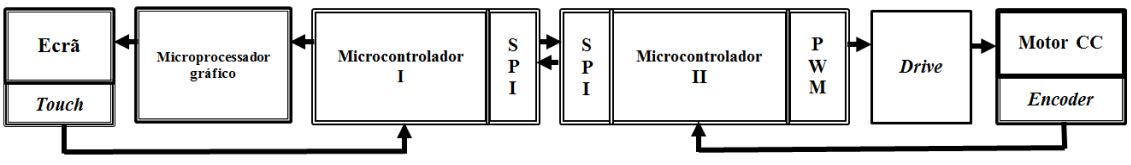


Figura 3.1 – Esquema base da solução desenvolvida baseada em microcontroladores

O microcontrolador I comunica com o ecrã através do microprocessador gráfico, que executa todas as tarefas gráficas ordenadas. Assim, implementou-se a

configuração correspondente à arquitetura com processamento gráfico externo, descrita no segundo capítulo (Figura 2.2).

A informação de comando proveniente da interface gráfica é enviada por SPI para o microcontrolador II da Figura 3.1 que efetua o controlo por PWM do motor. A posição angular e velocidade são obtidas a partir do *encoder* montado no eixo do atuador.

A parte seguinte do subcapítulo será dedicada ao *hardware* do protótipo. Serão abordados o microcontrolador, microprocessador gráfico e ecrã táctil para o módulo HMI, e para o outro módulo, de controlo, o segundo microcontrolador do protótipo, o motor CC com *encoder* e respetivo *drive*.

O microcontrolador I selecionado para o controlo da interface é um microcontrolador de 32 *bits* da Microchip, o PIC32MX795F512L (Figura 3.2) referido na documentação técnica dos seminários *Sagitron* 2012 e 2013, como bastante indicado em aplicações de desenvolvimento de interfaces gráficas *touch* com microcontroladores. Possui PMP em modo 16 ou 8 *bits*, e quatro módulos SPI. A Microchip comercializa-o integrado nas suas mais recentes soluções em *hardware*: PIC32 *Starter kit*, e *software* de desenvolvimento, designadamente biblioteca gráfica.

Toda a aprendizagem e desenvolvimento da componente gráfica base do projeto foi efetuada com o *kit* referido integrado com a placa MEB que inclui o microprocessador de imagem modelo SSD1226 da *Solomon Systech* (Figura 3.2).

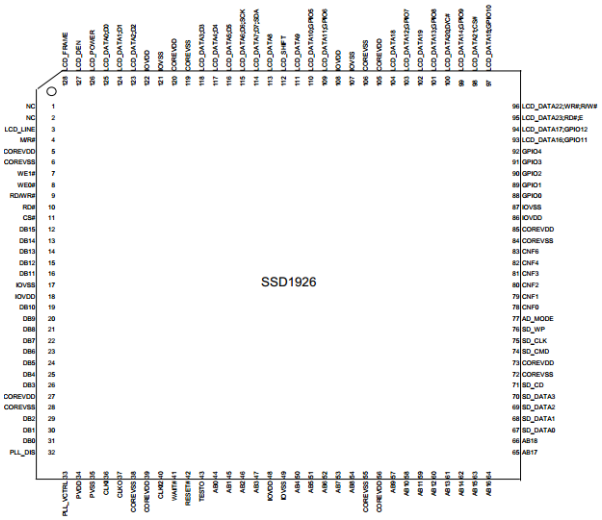


Figura 3.2 – Microprocessador de imagem SSD1226 *Solomon Systech*

A etapa de integração da HMI, e do seu *hardware* de suporte (placa MEB), na interface com microcontrolador II, permitiu testar a comunicação SPI em modo 8 *bits*, e identificar uma limitação em dois dos módulos de comunicação SPI, que é descrita no subcapítulo 3.5 do presente capítulo, assim como serão abordadas as ações alternativas tomadas. Refere-se por enquanto que foram necessárias a substituição da placa desenvolvimento (para a *Explorer* 16) e do microcontrolador de 32 *bits* por um de 16 *bits*: PIC24FJ256GA110, representado na Figura 3.3. De notar que o PIC24 inclui apenas 3 módulos de comunicação SPI que requerem uma prévia configuração por PPS (*Peripheral Pin Select*), e PMP 8 *bits* [12]. Este microcontrolador tem encapsulamento TQFP com 100 pinos, e inclui nos módulos SPI a possibilidade de efetuar transmissão de dados com 8 *bits*. Também dispõe de modos de funcionamento com gestão de consumo elétrico [13].

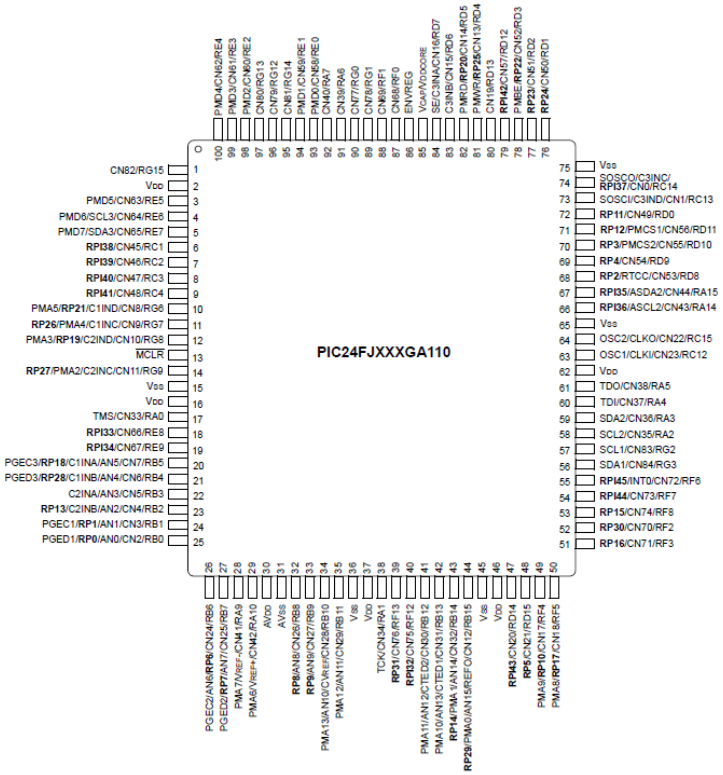


Figura 3.3 – O microcontrolador PIC24FJ256GA110

De seguida irá ser apresentado o ecrã táctil utilizado na implementação da HMI. Trata-se do MTF-TQ35SP811-AV da *Microtips Technology* (Figura 3.4).

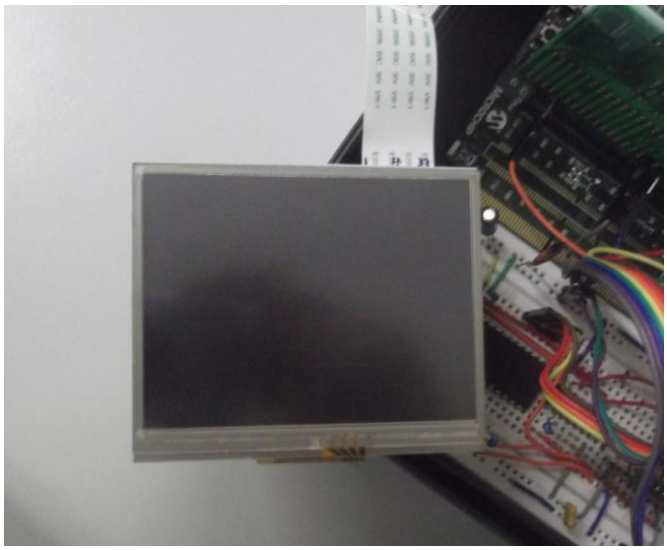


Figura 3.4 – Ecrã táctil MTF-TQ35SP811-AV

É um LCD TFT de 3.5’’com resolução QVGA (de 320x240 *pixels*), tratamento de superfície antirreflexo, potência retroiluminação de 456 mW e é capaz de operar em ambientes entre -20° C e 70° C.

A dimensão mínima do *frame buffer* para o *display* implementado no protótipo, com a profundidade de cor de 16 *bit*, é de 150 k*byte*. Este valor obtém-se a partir da equação 2.1 de dimensionamento (subcapítulo 2.2).

De seguida são enumeradas algumas características do ecrã na Tabela 3.1.

Tabela 3.1 – Resumo características elétricas do ecrã táctil MTF-TQ35SP811-AV

	Min.	Máx.
$\mathbf{V_{DD}}$	3.0 V	3.6 V
$\mathbf{I_{DD}}$	35 mA	45 mA

No que concerne ao *touch* resistivo, o manual refere apenas a sua fiabilidade, designadamente a capacidade de suportar no mínimo 1 milhão de toques com base num ensaio com uma caneta com extremidade com raio 0.8 mm e carga de 250 gf [14].

O microcontrolador selecionado para o controlo do motor elétrico de CC tem que possuir um módulo para controlo PWM, a possibilidade de comunicação por SPI e a interface para *encoder*.

O microcontrolador PIC18LF4431 de 40 pinos (Figura 3.5) e topologia PDIP (*Plastic Dual In-line Package*) caracteriza-se por ser da família de 8 *bits* da Microchip.

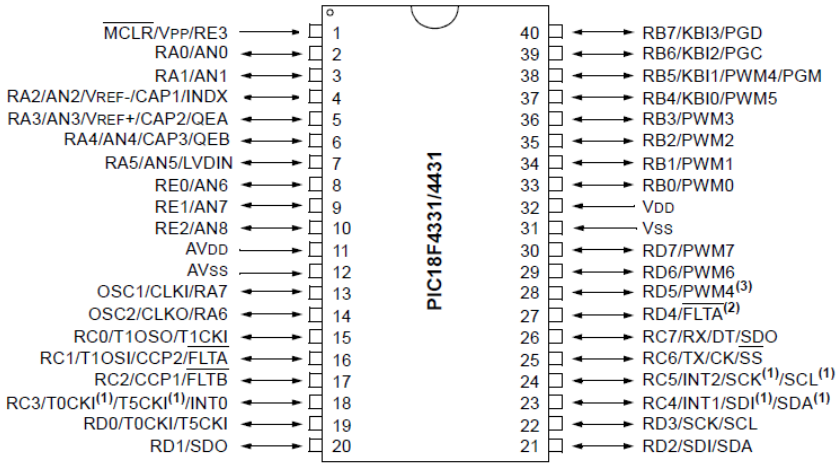


Figura 3.5 – Diagrama de pinos de entrada/saída do microcontrolador PIC18LF4431

A velocidade de processamento pode chegar a atingir 40 MHz através do uso de um cristal externo. No que concerne ao periférico, dispõe de módulo SSP (*Synchronous Serial Port*), que permite a comunicação com a interface gráfica desenvolvida, e módulo PWM com até três saídas complementares, frequência com resolução até 10 *bits*, *duty cycle* e período com resoluções até 14 *bits*. Inclui ainda no módulo MFM, o submódulo QEI. Este conjunto de características é adequado ao comando do motor em causa.

O QEI é um submódulo do MFM que permite a medição de posição e de velocidade e a identificação do sentido de rotação, através da descodificação da informação proveniente do *encoder*. É composto pelo controlo lógico QEI, o contador de posição e *postscale* da velocidade, representados no diagrama de blocos simplificado da Figura 3.6.

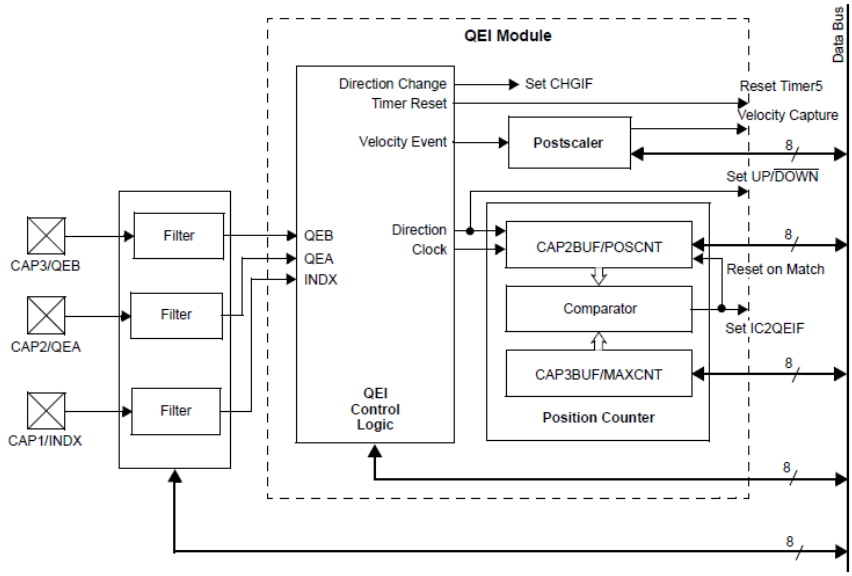


Figura 3.6 – Módulo QEI

A resolução da medição da posição depende do número de incrementos registados no POSCNT para a mesma amplitude rotacional. O QEI x2 e o QEI x4 são os dois modos de atualizar este registo e medir a posição.

Na Figura 3.7 estão representadas o número de transições por período para os sinais QEA.

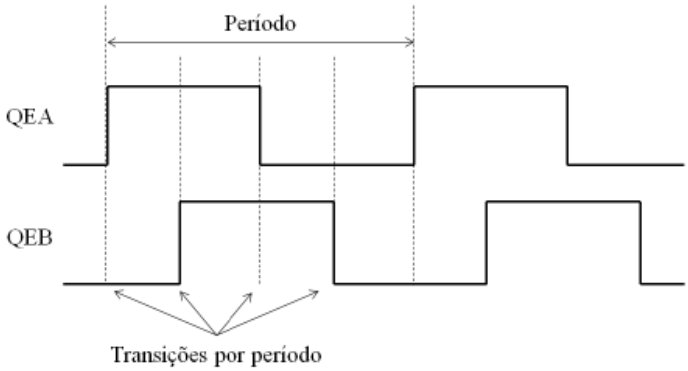


Figura 3.7 – Transições dos sinais QEA e QEB no modo QEI x4 [15]

Antes do início da descrição do motor CC com escovas empregado no protótipo, uma introdução sobre este acionamento elétrico.

O motor CC foi uma das primeiras máquinas desenvolvidas com o propósito de transformar energia elétrica em mecânica. A sua origem resultou da investigação efetuada por Michael Faraday, o investigador que formulou as bases do Eletro-

magnetismo. Esta ciência enuncia que se um fio condutor atravessado por uma corrente for inserido num campo magnético, desenvolver-se-á uma força que atuará sobre o condutor. A amplitude desta força depende da intensidade e orientação do campo e da corrente que flui no condutor e respetiva orientação.

Todos os motores CC com escovas (Figura 3.8) são constituidos pelos seguintes componentes: o estator, o rotor e um comutador.

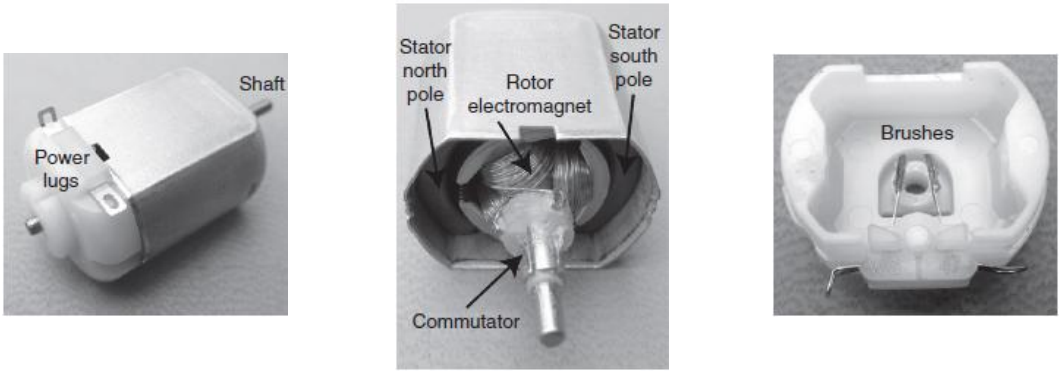


Figura 3.8 – Motor CC com escovas

O estator gera o campo magnético estático que envolve o rotor. O campo pode ser gerado por ímanes permanentes ou enrolamentos eletromagnéticos. Os vários tipos de motores podem distinguir-se através da forma construtiva do estator ou o modo de ligação dos enrolamentos à fonte. O rotor, também designado de armadura, é constituido por um ou mais enrolamentos, que quando excitado induz um campo magnético. Os pólos magnéticos do campo do rotor são atraídos pelo estator. Este efeito provoca a rotação do rotor. Durante a rotação do motor, os enrolamentos são constantemente excitados numa sequência diferente para que os pólos magnéticos do rotor não avancem relativamente aos pólos gerados pelo estator – comutação.

Nos motores CC com escovas, a comutação nos enrolamentos é mecânica. As escovas de grafite deslizam sobre o coletor, que está ligado a diferentes enrolamentos do rotor. A aplicação de uma tensão entre escovas induz um campo magnético dinâmico no seio do motor. As escovas e o coletor são os elementos constituintes do motor mais solicitados por atrito.

Os motores CC com escovas são utilizados numa vasta gama de aplicações, desde brinquedos, ventiladores, bancos de automóvel até máquinas industriais.

Este tipo de motores tem um custo reduzido, permitem ser controlados por *drives* de baixa complexidade tecnológica e consequentemente de baixo custo.

Na Figura 3.9 é possível visualizar o motor CC (da *Computer Optical Products*, *Inc*) utilizado no projeto. É um motor de ímanes permanentes dotado de *encoder* incremental e com as seguintes características obtidas experimentalmente devido à inexistência de catálogo do motor [15]:

- Tensão nominal: 24 V
- Corrente em vazio: 120 mA
- Velocidade em vazio: 4500 rpm
- Resolução do *encoder*: 400 ppr

Figura 3.9 – Motor CC com escovas [15]

O motor CC com escovas é conectado através de duas ligações com o respetivo *drive*.

A interface entre o *encoder* e o PIC18LF4431 ocorre através dos pinos representados na Tabela 3.2.

Tabela 3.2 – Ligações do *Encoder* ao PIC18

PINO	FUNÇÃO
6	QEB
5	QEA

Um motor CC necessita de uma alimentação elétrica (corrente/tensão) que não é compatível com a que pode ser disponibilizada por um microcontrolador. Assim é necessária a existência de um *drive* para fazer a interface entre os sinais de comando do microcontrolador e o motor.

Um desses *drives* passíveis de serem utilizados com motores CC de baixa potência é um circuito eletrónico que incorpora uma ponte H. Para a construção de uma ponte H são utilizados normalmente transístores que permitem o controlo bidirecional do motor a partir de quatro sinais digitais provenientes do microcontrolador. Na Figura 3.10 é apresentado o ST L298N, usado no projeto, e o respetivo esquema do circuito integrado que é constituído por duas pontes H.

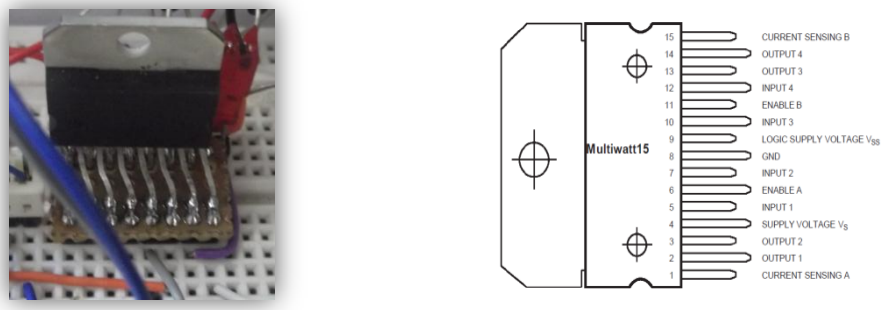


Figura 3.10 – Drive para motor CC: ST L298N da *STMicroelectronics*

Na implementação com o L298 fez-se uso de uma ponte H, permitindo controlar o motor nos dois sentidos.

A ligação entre o motor CC com escovas e o *drive* é efetuada através do *Output3*.

Na Tabela 3.3 é possível visualizar algumas das características elétricas sobre esta ponte de potência.

Tabela 3.3 – Resumo características elétricas do *drive* ST L298N

	Min.	Máx.
$\mathbf{V_S}$	V_{SS} + 2.5 V	46 V
$\mathbf{V_{SS}}$	4.5 V	7 V
$\mathbf{I_{out}}$	-	2 A
P	-	25 W

3.3 *Software*

Desenvolvido pela Microchip [16] o MPLABx é um *software* de desenvolvimento para aplicações com microcontroladores que corre nos sistemas operativos *Windows*, MAC e *Linux*. Inclui um editor pleno de recursos, *debugger*, gestor de projeto, simulador de *software* e é compatível com os programadores MPLAB ICD3, PM3 e PICkit 3 (Figura 3.11) que é o programador utilizado no projeto de interfaces gráficas.

Figura 3.11– O programador PICkit 3 da Microchip

A compilação para microcontroladores da família PIC32, PIC24 e PIC18 é efetuada através do MPLAB XC32, MPLAB XC16 e MPLAB XC8, respetivamente. No projeto foram efetuadas programação [17] em linguagem C e compilações com os três compiladores: XC32, XC16 para as interfaces gráficas e XC8 para o comando do motor CC.

A Microchip dispõe de uma ferramenta, com *download* gratuito, de *design* e desenvolvimento de ecrãs GUI (*Graphical User Interface*) para aplicações desenvolvidas em microcontroladores de 32 e 16 *bit*.

No presente projeto a ferramenta referida foi utilizada embebida e foi ativada a partir da janela *Tools* (Figura 3.12).

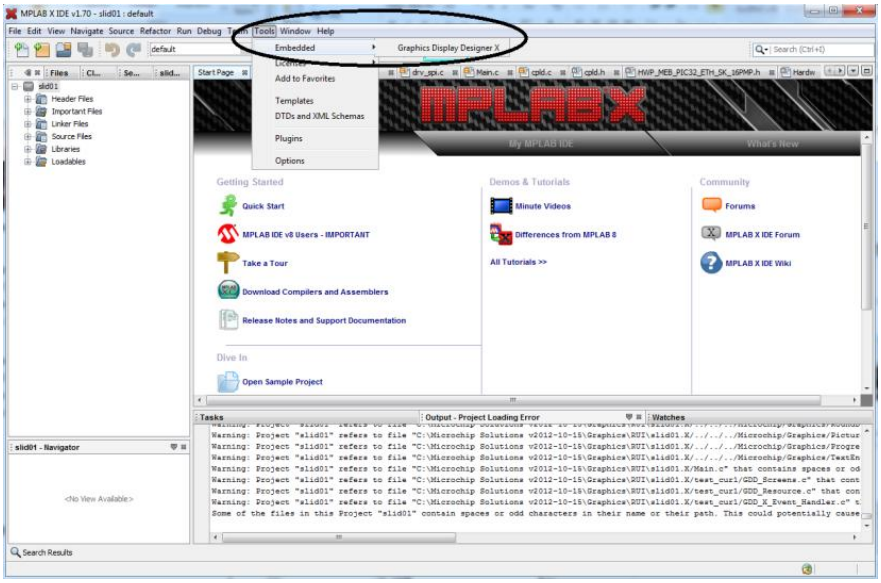


Figura 3.12 – Editor do *software* MPLABx com GDDx

Encontra-se representada na Figura 3.13 uma imagem do editor do GDDx.

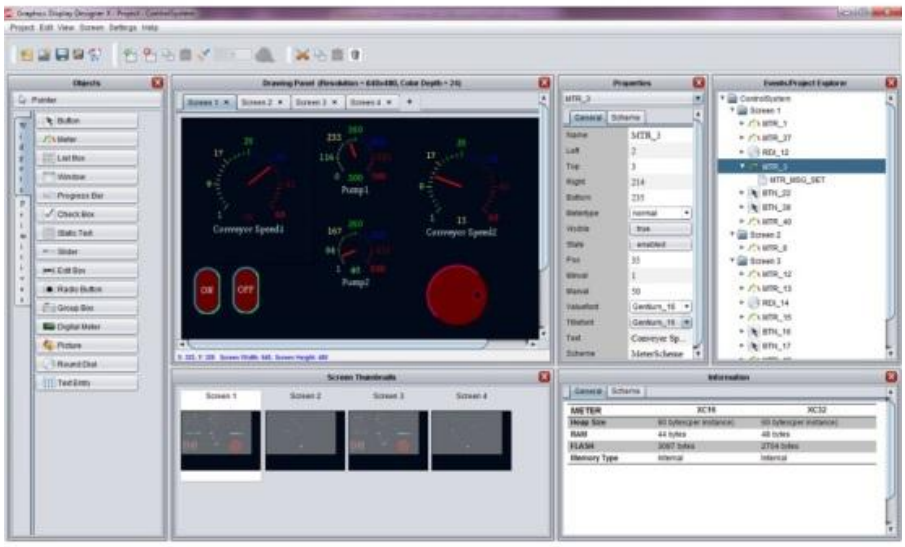


Figura 3.13 – Editor da aplicação informática GDDx

Esta aplicação é bastante intuitiva e dotada de inúmeros recursos, no entanto, de referir que, a experiência com a utilização dos *softwares* referidos durante a realização do projeto revelou ser conveniente encerrar primeiro a aplicação

GDDx e só depois o MPLABx. Outro aspeto que deve ser considerado é a impossibilidade de efetuar *undo* e/ou *redo*. As imagens inseridas no objeto *picture* devem já ter a dimensão definida com outra ferramenta, designadamente, *Paint-Net*, e o nome do ficheiro correspondente não pode ter espaços.

O *software* desenvolvido para MPLAB XC16 permite a comunicação entre o microcontrolador de 16 *bits* e dois elementos do projeto de *hardware*: o periférico interface gráfica e o microcontrolador de comando do motor CC. O código para MPLAB XC8 permite medir a posição do motor CC e gerar o sinal de comando do amplificador de potência.

As configurações principais dos registos de trabalho implementadas no código do PIC18LF4431 foram a definição do oscilador, *encoder* e filtros de sinal.

Definiu-se o oscilador para uma frequência de 8 MHz, de acordo com frequência do oscilador do PIC24FJ256GA110. Os *bits* 6 a 4 do registo OSCCON a 111: OSCCON = 0b01111100.

O módulo QEI associado ao *encoder* é ativado para modo posição e modo velocidade. No modo velocidade: o registo QEICON = 0b00101011, com modo de velocidade ativado: *bit* 7 igual a 0, e redução de 1:64 dos impulsos de velocidade: *bit* 1-0 com valor lógico 1.

No caso da posição os *bits* referidos tomam valor lógico 0.

É implementado um filtro através do registo DFLTCON que, segundo o manual técnico do microcontrolador, garante que os sinais de entrada só são admitidos após a obtenção de um sinal estável correspondente a três ciclos de amostragem consecutivos.

3.4 Implementação

O presente subcapítulo apresenta os vários elementos, anteriormente, referidos integrados no protótipo do sistema, a comunicação entre os microcontroladores utilizados e finalmente, estratégias para o controlo PWM e PID.

O sistema desenvolvido (Figura 3.14) é do tipo modular, e constituído pela placa HMI com *touch screen* e a placa de comando do motor CC. O microcontrolador da HMI e o de comando do motor comunicam entre si por uma ligação SPI.

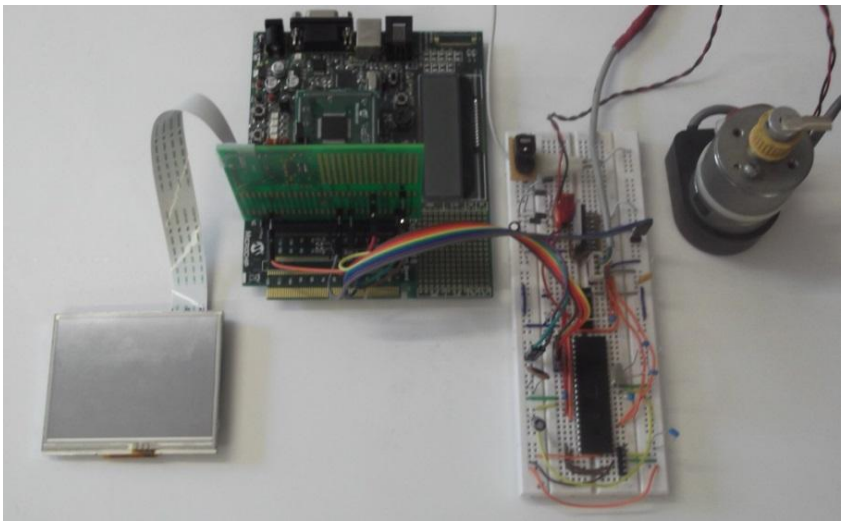


Figura 3.14 – A arquitetura global implementada para a HMI

A arquitetura do *hardware* da interface gráfica foi implementada numa placa de desenvolvimento *Explorer* 16 da Microchip, que alberga o microcontrolador PIC24, e conecta via *daughter board* com o módulo de *display* por *flat cable* (Figura 3.15 com moldura a azul escuro). O circuito de comando de potência, esta representado em conjunto com o motor CC com escovas na Figura 3.15 com moldura a azul claro.

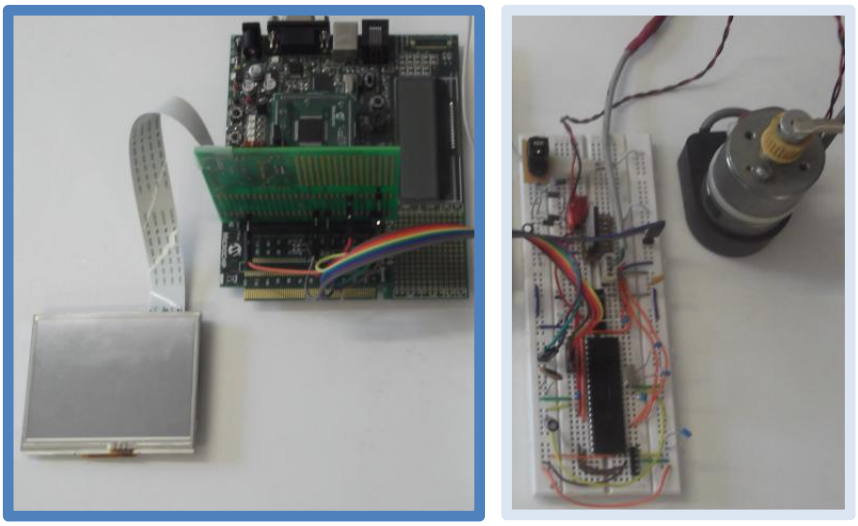


Figura 3.15 – A arquitetura implementada: módulos da interface gráfica e de comando de potência

As ligações para comunicação SPI entre o PIC24 e o PIC18 do módulo de comando do motor CC foram efetuadas por quatro fios através da *daughter board* referida.

O circuito eletrónico de comando do motor CC foi implementado de acordo com o esquema apresentado na Figura 3.16, utilizando o PIC18.

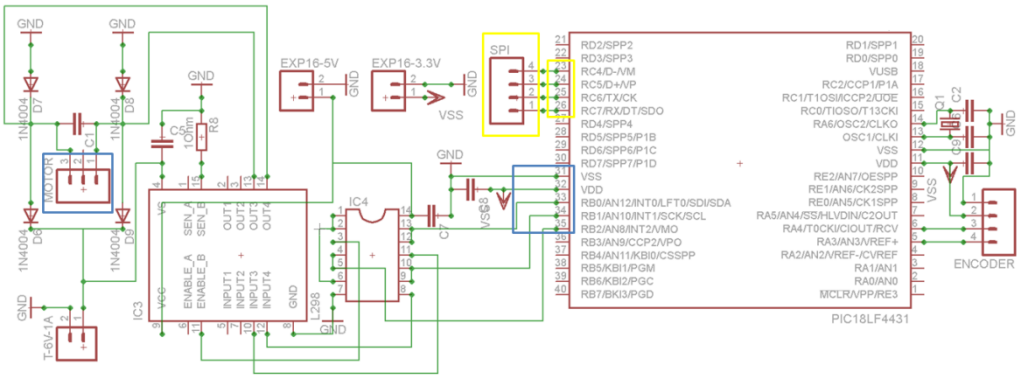


Figura 3.16 – Esquema elétrico desenvolvido para o comando do motor CC

Este é constituído por duas partes, uma relativa à comunicação SPI (Figura 3.16 com contorno a amarelo) e outra associada à ligação/comunicação com a ponte de potência L298N (Figura 3.16 com contorno a azul).

Foi utilizada uma placa *bread board* para a realização do circuito (Figura 3.17).

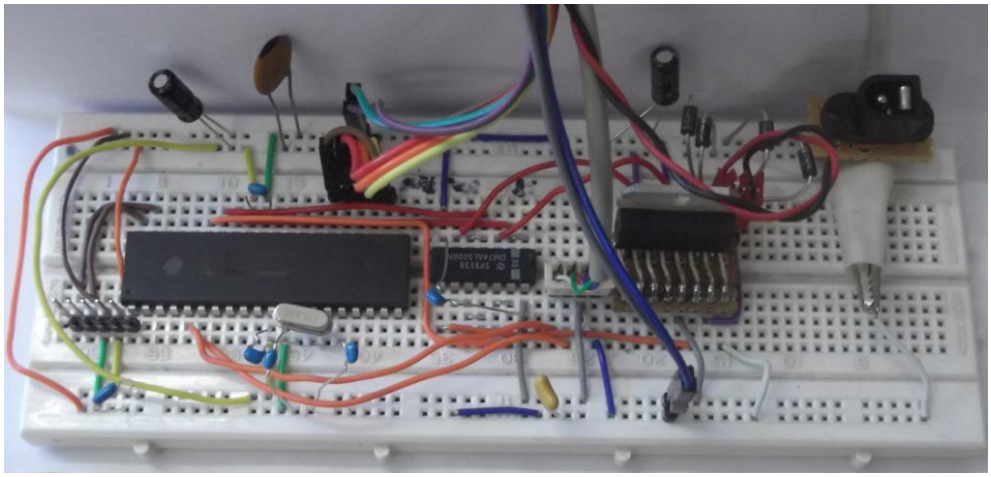


Figura 3.17 – Placa de comando do motor CC

O integrado NAND (Figura 3.18): DM74ALS00AN da *Fairchild Semiconductor*, foi utilizada para compatibilizar o nível de tensões entre o microcontrolador e o *drive* (L298N).

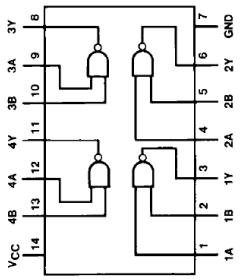


Figura 3.18 – Porta lógica NAND DM74ALS00AN

Na criação do projeto de *software* associado ao ecrã a implementar, o programa de desenvolvimento incorpora automaticamente o código correspondente, através da seleção do *kit* de desenvolvimento, nas definições do ficheiro *hardwareprofile.h* (Figura 3.19). As portas no *hardware* para comunicação ficam assim pré-definidas.

```
    //#include "Configs/HWP_GFXv3_PIC_SK_16PMP_WQVGAv1.h"

/*********************************************************************
 * Hardware Configuration for
 * Starter Kit
 * MultiMedia Development Board
 * Display TFT-G240320LTSW-118W-E
 ********************************************************************/
//#include "Configs/HWP_MEB_PIC32_STK_8PMP.h"
//#include "Configs/HWP_MEB_PIC32_USB_SK_8PMP.h"
//#include "Configs/HWP_MEB_PIC32_ETH_SK_8PMP.h"

//#include "Configs/HWP_MEB_PIC32_GP_SK_16PMP.h"
//#include "Configs/HWP_MEB_PIC32_USB_SK_16PMP.h"
#include "Configs/HWP_MEB_PIC32_ETH_SK_16PMP.h"

//#include "Configs/HWP_MEB_dsPIC33E_SK_8PMP.h"

/*********************************************************************
 * Hardware Configuration for
 * Low Cost Controllerless (LCC) Daughter Board
```

Figura 3.19 – Ficheiro de configuração do *hardware*

Para a comunicação com o controlador do motor CC optou-se pela comunicação SPI por ser de fácil implementação, necessitando apenas de um cabo com quatro condutores, e os microcontroladores selecionados para o projeto estão dotados de

módulos SPI. No comando do motor CC utilizou-se uma aplicação típica *master*/*slave*.

Na comunicação série a transferência de informação ocorre *bit* a *bit* [10]. O exemplo da Figura 3.20 mostra o envio de um *byte*, *e.g.*, 10011101.

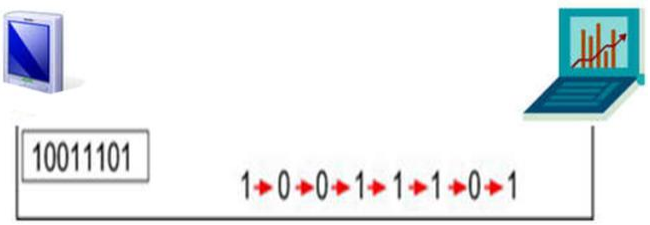


Figura 3.20 – Comunicação série

Com o propósito de efetuar o mapeamento das portas disponíveis para a comunicação SPI, efetuou-se a análise dos manuais referentes à placa de desenvolvimento Microchip MEB, PIC32 ESK e ao PIC32MX795F512L e implementou-se a ligação entre os microcontroladores PIC32 e PIC18 através da barra de 28 pinos (J5) presente na MEB e assinalada na Figura 3.21. Foi assim possível testar os módulos 2A e 3A da comunicação SPI [18].

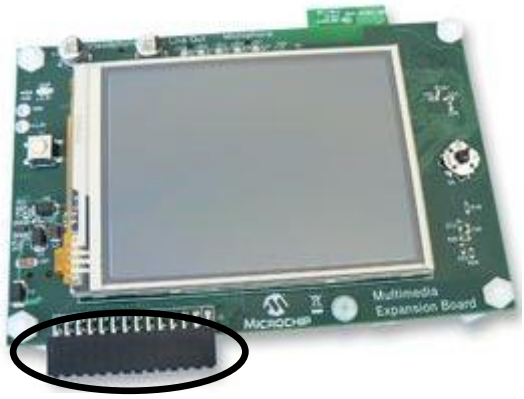


Figura 3.21 – MEB (J5)

Foi programado o código referente à inicialização (Figura 3.22) e à geração de impulsos de *clock* e envio de um carater.

```
void initspi (void) {

char junk;

SPI2ACONbits.ON = 0;              //desabilitar SPI para reset
junk = SPI2ABUF;                  // lê o buffer SPI para limpar o buffer receção
SPI2ABRG = 100;                   // [(f_clk/f_spi)/2]-1=BRG
SPI2ACONbits.MSTEN = 1;   //habilita master mode
SPI2ACONbits.CKE = 0;             // define "clock-to-data timing"
SPI2ACONbits.CKP = 0;
SPI2ACONbits.ON = 1;              // liga SPI
      }
```

Figura 3.22 – Código de inicialização do SPI2A

Da análise no osciloscópio não foi possível definir sinal de saída para o módulo SPI2A, confirmando-se através do multímetro a inexistência de continuidade no pino referente ao SDO2A, visto a informação não chegar ao microcontrolador.

De seguida, com recurso ao código em tudo análogo ao usado para o módulo SPI2A verificaram-se através do osciloscópio os sinais correspondentes ao módulo SPI3A (Figura 3.23), sendo representados: a azul claro o *clock* para comunicação a 8 *bit* e a amarelo a saída (SDO) referente à letra 'R' enviada pela MEB.

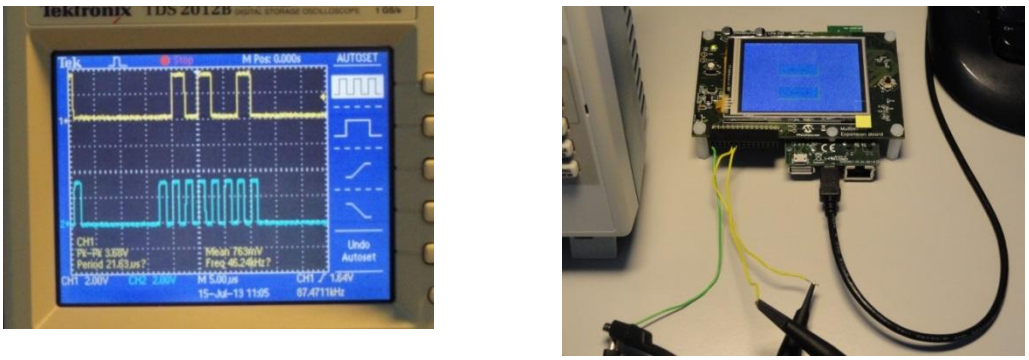


Figura 3.23 – Análise no osciloscópio para SPI3A

Configurou-se a MEB segundo a Tabela 3.4 com as disponibilidades de pinos para a comunicação com o microcontrolador de controlo do motor. Verificou-se a inexistência de sinal no pino para SDI nos dois módulos.

Tabela 3.4 – Pinos disponíveis para a comunicação SPI (MEB e ESK PIC32)

Módulo SPI	Pino	Função
SPI2A	11	SCK
SPI3A	18	SDO
	20	SCK

O sinal SDI é essencial à receção de dados de entrada na MEB e, consequentemente, no microcontrolador mestre, inviabilizando a comunicação bidirecional com o exterior da interface gráfica. O problema foi reportado à Microchip, que não nega a existência deste erro, nem indica uma solução congénere implementável. Foi necessário optar por uma nova placa de desenvolvimento para a HMI e isto numa fase em que era necessário testar o código desenvolvido com a biblioteca gráfica, interligando os dois subsistemas: as partes relativas ao motor e à HMI. A opção passou por selecionar com base em critérios como a manutenção dentro da gama de *hardware* definidos na biblioteca gráfica: EXPLORER_16, PIC24FJ256DA210_DEV_BOARD, MEB_BOARD e PIC_SK, e rapidez na obtenção deste recurso.

A escolha recaiu na placa *Explorer* 16. É de referir que esta placa de desenvolvimento também pode incluir um PIC32, mas a obtenção deste componente e o respetivo PIM (*Plug-in Module*) poderia atrasar ainda mais os *deliverables* definidos no início do projeto, pelo que se incluiu na placa referida um microcontrolador com 16 *bits*, o PIC24, disponível em laboratório.

Na implementação com a placa *Explorer* 16 e o microcontrolador PIC24 já foi possível definir todos os pinos necessários à comunicação SPI de acordo com a Tabela 3.5. e disponíveis no *socket* da placa.

Tabela 3.5 – Pinos alocados à comunicação SPI1

Pino	Função
88	CS
68	SCK
25	SDO
24	SDI

Foi utilizado o módulo SPI1 no PIC24 e o módulo SSP no PIC18.

A comunicação entre a interface gráfica e a interface de comando do motor é efetuada através de quatro fios (amarelo, laranja, vermelho e castanho) que ligam o microcontrolador de 8 *bits* (vértice inferior esquerdo da Figura 3.24) da placa de comando do motor e a *daughter board* do tipo *display adapt board* da marca E*tconcep*, representada na Figura 3.24 perpendicularmente à placa HMI. Esta alberga o microcontrolador de 16 *bits* (assinalado a branco na Figura 3.24).

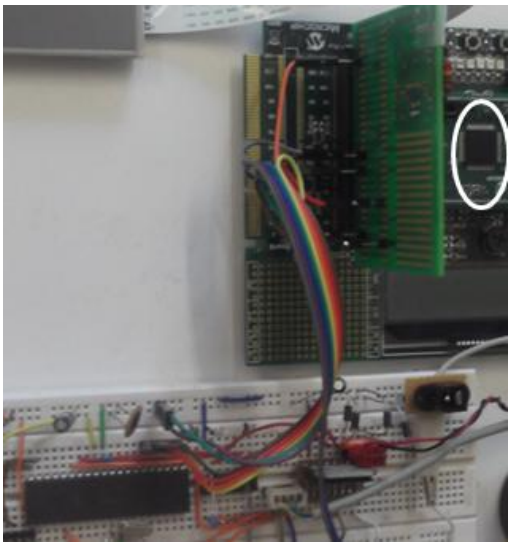


Figura 3.24 – Ligações para comunicação SPI entre placas HMI e de comando do motor Corrente Contínua

A comunicação SPI entre os dois módulos do protótipo foi caracterizada, iniciando-se de seguida a apresentação das técnicas de controlo implementadas.

O PWM é uma técnica de controlo em que o impulso é gerado em intervalos regulares (T), de acordo com o registo de período de uma função *timer*. A largura de impulso (T_{on}) não é fixa, sendo possível definir, por programação, o seu valor percentual relativo ao período T (Figura 3.25) [19]. O *duty cycle* é a fração do período em que o impulso ou degrau é máximo [3] e T_{on}.

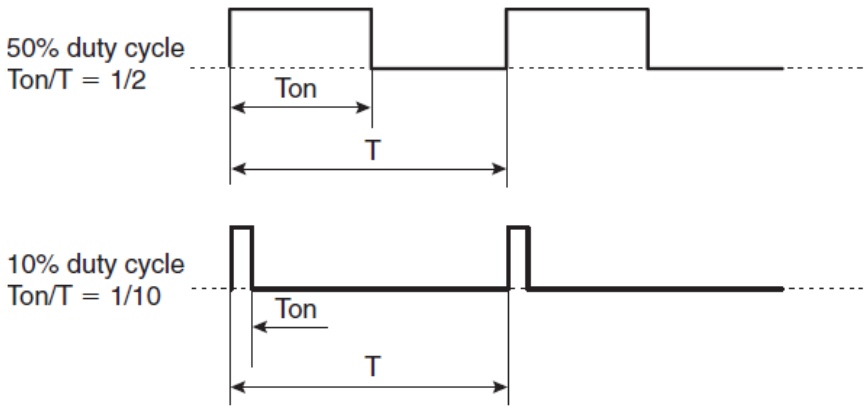


Figura 3.25 – PWM com diferentes *duty cycle*

Assim, há duas situações limite para o *duty cycle*: 0 e 100%. No primeiro caso o sinal está OFF e no segundo sempre ON.

Dois sinais de PWM gerados de forma complementar (Figura 3.26) possibilitam o movimento do motor em diferentes sentidos e velocidades de rotação.

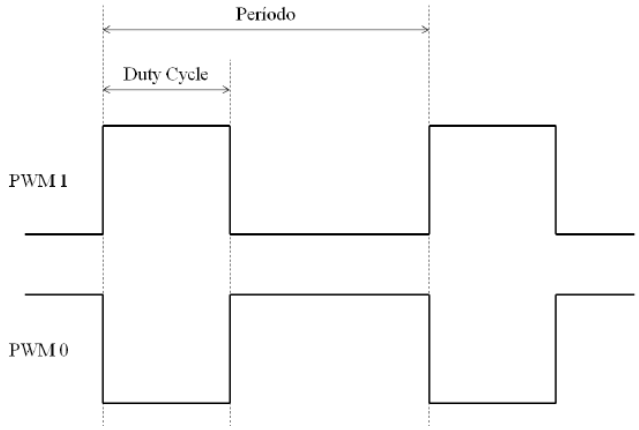


Figura 3.26 – PWM gerado em modo complementar [15]

A velocidade do motor sem carga é proporcional à tensão aplicada aos seus terminais. Assim, a variação da tensão permite o controlo do motor.

O PWM é utilizado para a implementação do comando da velocidade conforme a Figura 3.27.

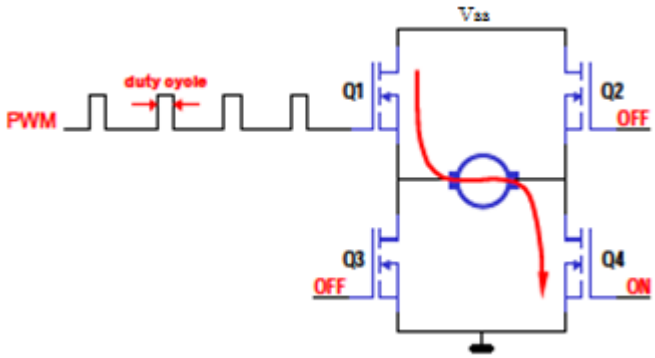


Figura 3.27 – Comando velocidade por PWM

Na Tabela 3.6 estão representadas as ligações entre o módulo de PWM do microcontrolador do PIC18 e o *drive* do motor CC.

Tabela 3.6 – Ligações do PWM do PIC18 ao *Drive*

Drive	Microcontrolador	
PINO	PINO	FUNÇÃO
Enable B	35	PWM2
Input3	34	PWM1
Input4	33	PWM0

O microcontrolador PIC18 calcula e disponibiliza o sinal adequado de comando ao motor com base na informação de entrada indicada pelo utilizador e o *feed-back* da grandeza angular de saída. Neste trabalho é utilizado, e parcialmente implementado no módulo de controlo do motor, um algoritmo do tipo PID pois é bastante comum em aplicações industriais e relativamente fácil de implementar.

O modelo do sistema em tempo contínuo encontra-se na Figura 3.28.

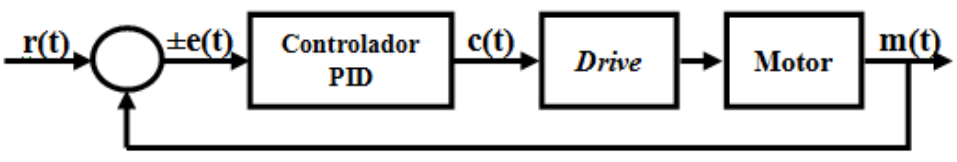


Figura 3.28 – Modelo do sistema

A referência é r(t), sendo m(t) a resposta medida, e(t) o erro e c(t) a resposta do controlador.

Na implementação de um controlador PID, existem três termos baseados no erro de medição:

- $k_P e(t)$ é *o termo proporcional*
- $k_I \int_0^t e(t)dt$ é *o termo integral*
- $k_D \frac{de(t)}{dt}$ é *o termo derivativo*

O sinal de controlo corresponde então à soma dos três termos (Equação 3.1):

$$c(t) = k_P e(t) + k_I \int_0^t e(t)dt + k_D \frac{de(t)}{dt} \qquad \text{Equação 3.1}$$

Aplicando as transformadas de Laplace à Equação 3.1 obtém-se a função transferência (Equação 3.2).

$$\frac{C(S)}{E(S)} = k_P + \frac{k_I}{S} + k_D S \qquad \text{Equação 3.2}$$

O modelo do sistema com controlador digital está representado na Figura 3.29.

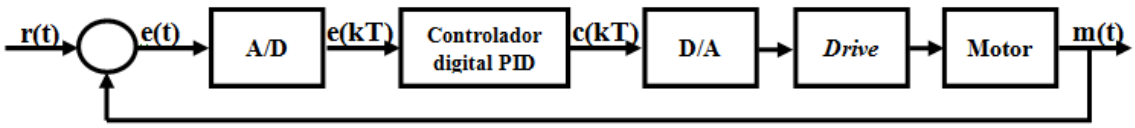


Figura 3.29 – Modelo do sistema com controlador digital

Para implementar o controlador PID no sistema digital, o microcontrolador terá que efetuar aproximações para os termos integral (Equação 3.3) e derivativo (Equação 3.4).

$$\int_0^t e(t)dt \sim T \sum_{h=1}^{k} \frac{e[(h-1)T] + e(hT)}{2} \qquad \text{Equação 3.3}$$

$$\frac{de(t)}{dt} \sim \frac{e(kT) - e[(k-1)T]}{T} \qquad \text{Equação 3.4}$$

Assim, com as aproximações referidas, a equação 3.1 pode ser reescrita para tempo discreto, obtendo-se o algoritmo de posição (Equação 3.5):

$$c(kT) = K\left\{e(kT) + \frac{T}{T_i}\sum_{h=1}^{k}\frac{e[(h-1)T] + e(hT)}{2} + \frac{T_d}{T}\left[e(kT) - e[(k-1)T]\right]\right\} \qquad \text{Equação 3.5}$$

Aplicando a transformada z à equação 3.5 obtém-se a equação 3.6.

$$\frac{C(z)}{E(z)} = k_P + \frac{k_I}{1 - z^{-1}} + k_D(1 - z^{-1}) \qquad \text{Equação 3.6}$$

onde $k_P = K - \frac{k_I}{2}$, $\quad k_I = \frac{KT}{T_i}$, $\quad k_D = \frac{K\,T_d}{T}$

O algoritmo de velocidade (Equação 3.7) efetua o cálculo da variação do sinal de saída do controlador relativamente ao instante imediatamente anterior [20].

$$\Delta c(kT) = k_P\left[e(kT) - e[(k-1)T]\right] + k_I e(kT) + k_D\left[e(kT) - 2e[(k-1)T] + e[(k-2)T]\right] \qquad \text{Equação 3.7}$$

Considerando:

$$e(kT) = r[(k-1)T] + r[(k-2)T] - m(kT) \qquad \text{Equação 3.8}$$

e aplicando a transformada z à Equação 3.7 obtém-se (Equação 3.9):

$$C(z) = -k_P M(z) + k_I\frac{R(z) - M(z)}{1 - z^{-1}} - k_D(1 - z^{-1})M(z) \qquad \text{Equação 3.9}$$

Este algoritmo possui algumas vantagens relativamente ao algoritmo de posição, designadamente, não ser necessária a inicialização quando se comuta o modo de operação de automático para manual [21] e protege contra a possibilidade de *wind-up*; no algoritmo de posição o somatório do erro no termo integral poderá provocar a saturação do controlador [20].

A Microchip (na nota técnica AN718) propõe um fluxograma para algoritmo PID cujo extrato se encontra representado na Figura 3.30.

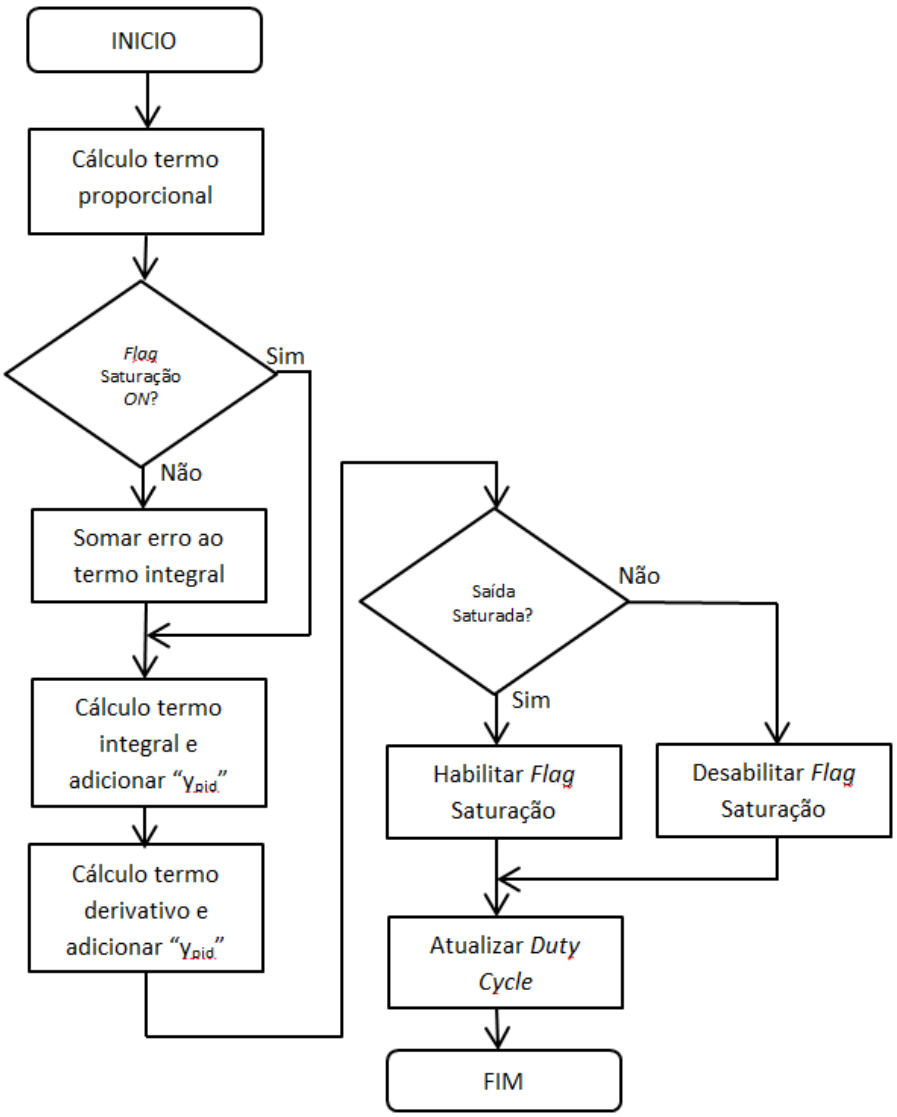


Figura 3.30 – Fluxograma algoritmo PID

No projeto foi implementada a ação proporcional de controlo do algoritmo de posição, e inclui suporte gráfico na HMI para as três componentes de controlo PID.

3.5 Conclusão

Os componentes da arquitetura do protótipo desenvolvida originalmente foram revistos em virtude de alguns contratempos, mas o resultado final enquadra-se no esquema base desenvolvido, mantendo-se a configuração "com microprocessador gráfico externo". Foram descritos os elementos principais constituintes da arquitetura: microcontroladores, ecrã táctil, motor CC com escovas, *encoder*, *drive* do

atuador, e foram abordados os módulos comunicação SPI e controlo PWM e submódulo QEI utilizados para implementação da HMI com controlo de motor CC. A abordagem teórica elementar ao controlo PID é enquadrável como base para desenvolvimento das interfaces gráficas, para o ajuste dos parâmetros de controlo por PID de posição e velocidade do veio do motor CC com escovas, conforme descrito no capítulo seguinte.

4 Interface gráfica

4.1 Introdução

Após a apresentação do protótipo com HMI táctil de comando de motor CC, o capítulo presente é dedicado à apresentação dos ecrãs da HMI.

A descrição detalhada das interfaces criadas é precedida pela caracterização das API necessárias ao desenvolvimento e implementação dos objetos virtuais, completando-se assim os elementos base para o desenvolvimento com a biblioteca gráfica e o GDDx da Microchip, abordados nos subcapítulos 2.4 e 3.4.

4.2 Desenvolvimento

O *software* MPLABX IDE e a aplicação GDD*x* permitem o desenvolvimento de programas para integração de interfaces gráficas em aplicações de comando de um motor. A ferramenta GDDx, embora bastante abrangente, não inclui a totalidade dos recursos da biblioteca gráfica Microchip. Assim, tem que se programar diretamente com a biblioteca para o desenvolvimento de soluções para além do domínio de abrangência do GDDx da Microchip, como, por exemplo, a implementação de uma máquina de calcular táctil. Para o *text entry* gerado automaticamente, o número máximo de botões é oito (Figura 4.1).

Figura 4.1 – O objeto *text entry* implementado a partir do GDDx

As interfaces são constituídas por objetos: *widgets*, que podem ser criados a partir de modelos pré-definidos ou gerados de raiz.

Para conceber um novo objeto terão que ser criados e/ou alterados os ficheiros representados na Figura 4.2.

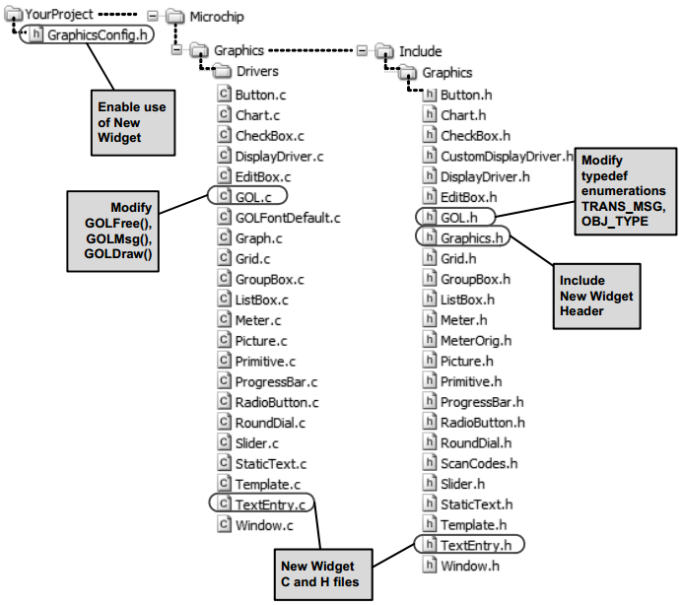


Figura 4.2 – Ficheiros da biblioteca gráfica associados à criação de um objeto novo

A biblioteca gráfica reúne em listas os objetos ativos (Figura 4.3 **a**) e em modo de receção de mensagens. As funções *GOLDraw* e *GOLMsg* exercem as suas ações (Figura 4.3 b) com base na lista referida.

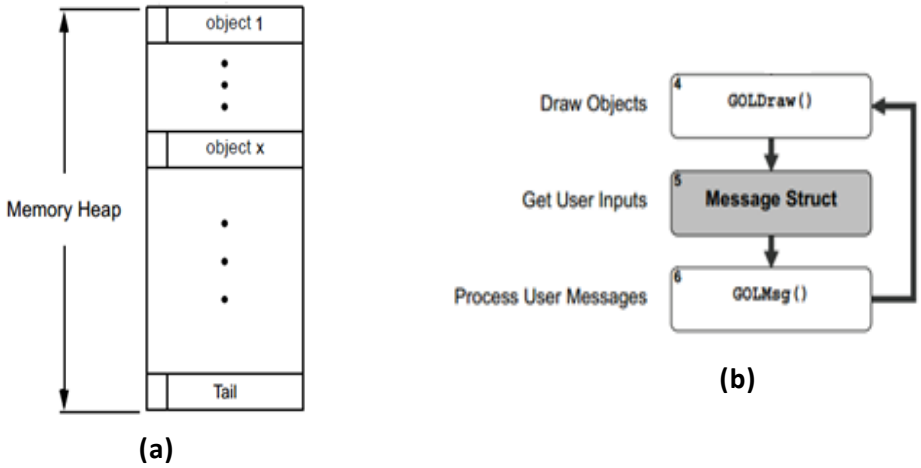


Figura 4.3 – Organização da pilha de objetos base da ação das funções *GOLDraw* e *GOLMsg* da biblioteca gráfica

No desenvolvimento da interface gráfica, cada objeto criado é adicionado à respetiva lista, correspondente ao seu menu. Somente uma lista de objetos pode estar ativa em cada instante.

O desenho dos objetos é efetuado pela GOL que analisa o estado de cada objeto na lista ativa. A sequência de desenho é do topo para a base da lista.

As API para implementação dos elementos virtuais, podem ser chamadas no modo NONBLOCKING através da máquina de estados representada na Figura 4.4.

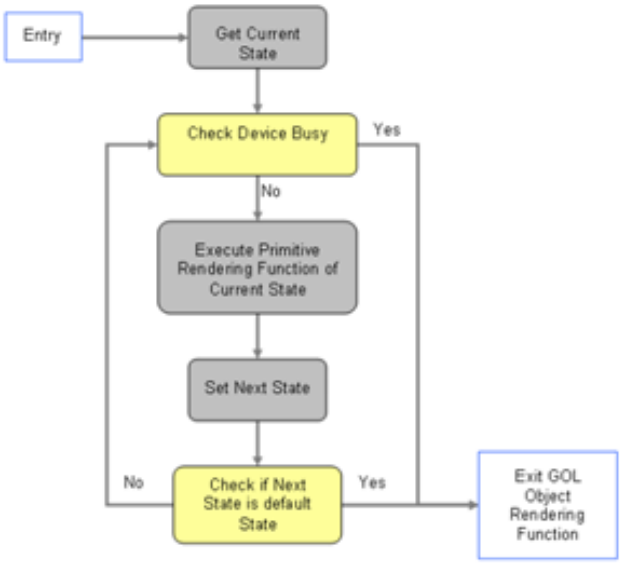


Figura 4.4 – Diagrama de estados no modo NONBLOCKING

Sempre que é executado um passo na conceção dos objetos, o estado do respetivo elemento pictórico é atualizado e se não estiver concluído, a ação associada pode ser finalizada posteriormente, libertando assim o MCU para execução de outras tarefas.

O esquema representado na Figura 4.5 ilustra como utilizar a biblioteca gráfica e as suas API.

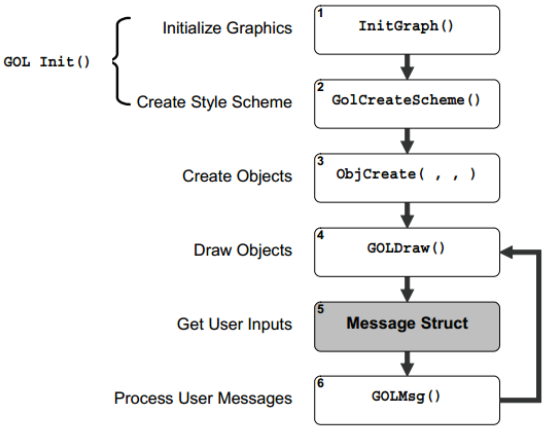


Figura 4.5 – Estrutura da biblioteca gráfica

As cinco funções base das APIs referidas são descritas de seguida, iniciando a descrição com a *GolCreateScheme*, através do código para definição de cores que deve ser estruturado de acordo com o apresentado na Figura 4.6. É de notar a utilização de um apontador para a GOL_SCHEME, que é posteriormente atribuído à função *GOLCreateScheme*(). Neste exemplo define-se a cor 0 do *widget* como preto e a cor 1 como azul, código RGB: (0,0,255).

```
GOL_SCHEME* altScheme;                  // declaração de estilo
// tipo de estilo

altScheme = GOLCreateScheme ();         // criação estilo alternativo

altScheme-> TextColor0 = BLACK;         // cor_0 de texto
altScheme-> TextColor1 = BRIGHTBLUE;    // cor_1 de texto
```

Figura 4.6 – Código de programação para definição de cores de um *widget*

A API *"ObjCreate"* é uma designação genérica para os possíveis objetos virtuais: *widgets*, criados.

Neste código exemplo para definição de *widget* botão (Figura 4.7) define-se um apontador tipo BUTTON, atribuindo a variável à função *BtnCreate*(x,x,x,x,x,x,x,x,x,x).

Figura 4.7 – *Widget* do tipo botão

```
BUTTON *pBTN_1;
   pBTN_1 = BtnCreate(  BTN_1,                  //nome
                        211,                    //esquerda
                        5,                      //topo
                        271,                    //direita
                        36,                     //base
                        0,                      //concordância
                        BTN_DRAW,               //estado
                        NULL,                   //imagem
                        (XCHAR*)uno_BTN_1text,  //texto: DOWN
                        defscheme               //scheme
                     );
```

Figura 4.8 – Código de programação para definição de *widget* do tipo botão

Esta função (Figura 4.8) é constituída por dez parâmetros que representam, respetivamente, o nome, a posição do *widget* relativamente ao vértice superior esquerdo do ecrã, se os vértices do botão são retos ou admitem concordância, o estado do botão: ativo ou não ativo, se admite imagem, texto e o tipo de esquema de cores.

O *widget* do tipo *slider* (Figura 4.9) é definido de forma análoga ao botão, mas com a diferença entre o campo referente ao estado e ao esquema de cores, em que se define a gama, resolução e posição inicial, conforme Figura 4.10.

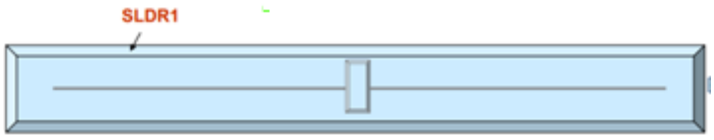


Figura 4.9 – *Widget* do tipo *slider*

```
SLIDER *pSLD_1;
   pSLD_1 = SldCreate(  SLD_1,                          //nome
                12,                                     //esquerda
                5,                                      //topo
                143,                                    //direita
                44,                                     //base
                SLD_DRAW | SLD_DRAW_THUMB,              //estado
                100,                                    //gama
                5,                                      //resolução
                50,                                     //posição inicial
               defscheme                                //scheme
             );
```

Figura 4.10 – Código de programação para definição de *widget* do tipo *slider*

A função *GOLDraw*() implementa o desenho de objetos e a função *GOLMsg*() processa/transfere a mensagem que define a ação e gera a alteração desejada no *widget*. No código apresentado na Figura 4.11, estão representados em itálico exemplos das funções referidas no interior de um ciclo *while*.

```
while(1){
if (GOLDraw()){          // Função desenha objeto GOL
TouchGetMsg(&msg);   // Recolhe mensagem do touch screen
GOLMsg(&msg);          // Processa mensagem
}
}
```

Figura 4.11 – A *GOLDraw*() e *GOLMsg*() implementadas no ficheiro *main*.c

A função *GOLMsgCallback*(x,x,x) utiliza as mensagens transferidas para implementar uma ação. Esta ação independentemente de se reportar a um sistema ou *widget* é baseada em eventos singulares. A *GOLMsgCallback*(x,x,x) é chamada pela função *GOLMsg*(). Na Figura 4.12 é apresentado um fluxograma exemplo para acionamento ON/OFF.

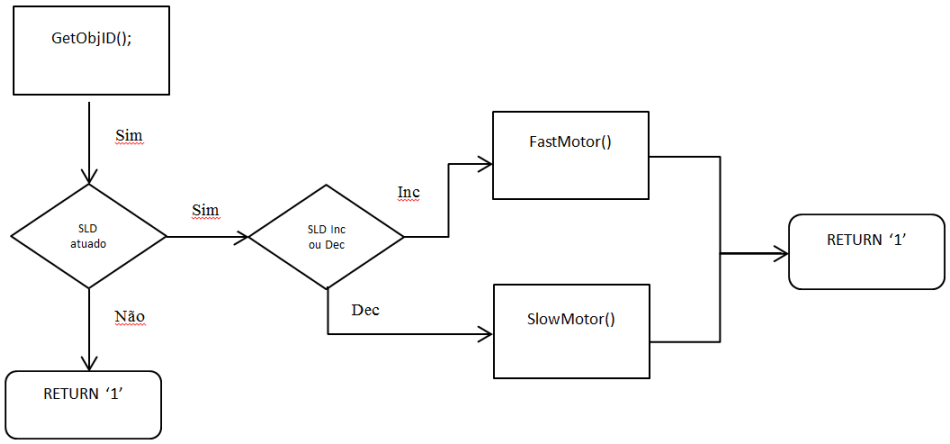


Figura 4.12 – Fluxograma exemplo para acionamento ON/OFF

A Figura 4.13 mostra um fluxograma para implementação da variação da velocidade de motor.

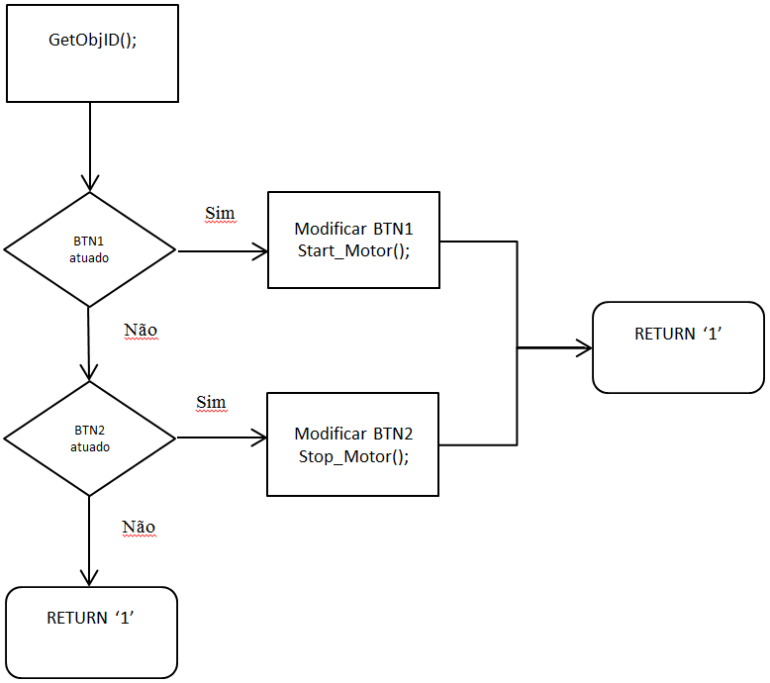


Figura 4.13 – Fluxograma exemplo para implementação da variação da velocidade do motor

É apresentado no anexo A o código exemplo com dois botões e *slider* para a função *GOLMsgCallback*().

O código gerado pela aplicação GDDx não remove o código diretamente inserido pelo programador, no interior desta função.

A função *GOLDrawCallback*() é chamada pela *GOLDraw*() e não usa diretamente as mensagens transferidas; as ações do sistema ou *widget* são baseadas em eventos contínuos. É a única função onde a alteração das propriedades dos elementos virtuais implementada pelo programador é prioritária relativamente a qualquer predefinição do *software*.

No anexo A está representado o código associado à implementação de barras indicadoras da variação de *slider* (Figura 4.14) para a função *GOLDrawCallback*().

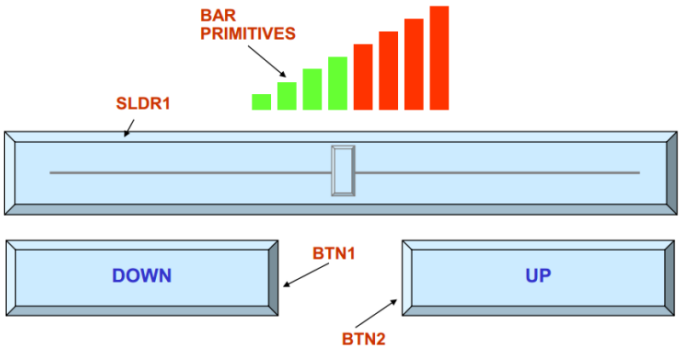


Figura 4.14 – Exemplo com barras indicadoras da variação slider

De modo análogo ao que ocorre com a função *GOLMsgCallback*() toda a programação efetuada dentro desta API não é alterada pelas atualizações protagonizadas pela ferramenta GDDx da Microchip.

4.3 Programação

Depois de enumeradas as funções que serão utilizadas, a sua estrutura e requisitos, procede-se à programação do microcontrolador.

O *hardware* utilizado inicialmente para a programação e onde se desenvolveu grande parte do trabalho foi a MEB e PIC32 ESK da Microchip. (Figura 4.15), devido à inclusão de um PIC32 que dispõe de características importantes em aplicações gráficas já indicadas nos subcapítulos 2.2 e 3.3.

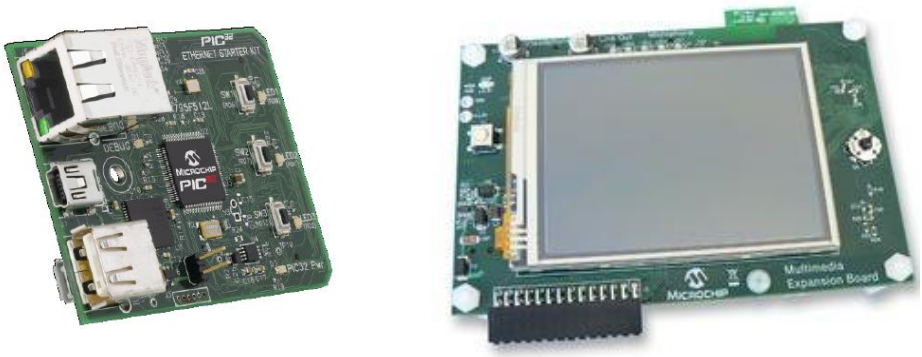


Figura 4.15 – Placas de desenvolvimento PIC32 ESK e MEB

Obtiveram-se vários resultados na componente gráfica do projeto, dos quais são representados exemplos nas Figura 4.16 a Figura 4.21.

Figura 4.16 – Ecrãs desenvolvidos com PIC32: "Ajuda" e "Inicial"

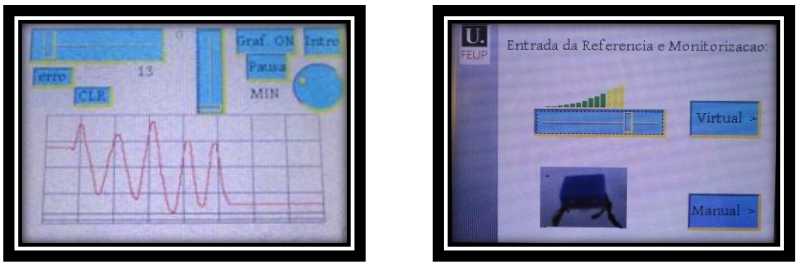


Figura 4.17 – Ecrãs desenvolvidos com PIC32: "Monitorização gráfica" e "Introdução à implementação"

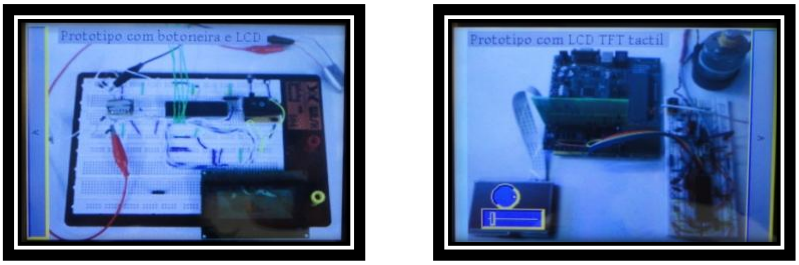


Figura 4.18 – Ecrãs desenvolvidos com PIC32: "Implementação I" e "Implementação II"

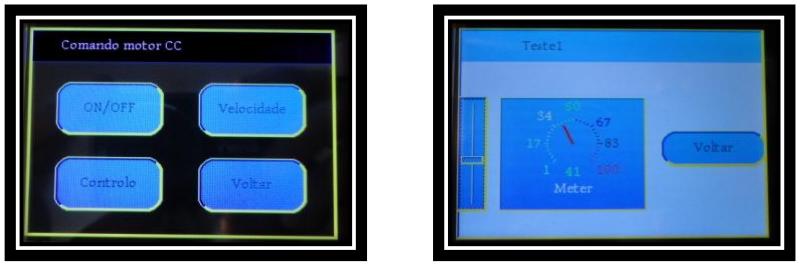


Figura 4.19 – Ecrãs desenvolvidos com PIC32: "Comando" e "Contador analógico"

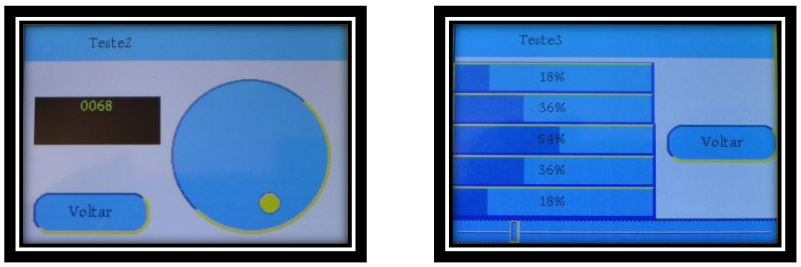


Figura 4.20 – Ecrãs desenvolvidos com PIC32: "Contador digital" e "Contador barras"

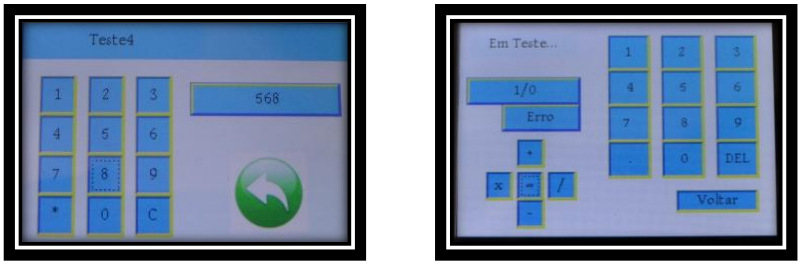


Figura 4.21 – Ecrãs desenvolvidos com PIC32: "*Keypad*" e "Calculadora"

Entretanto foram substituídos o microcontrolador e o respetivo suporte de *hardware* e alterado o compilador, pelas razões já referidas no relatório.

Toda a programação desenvolvida baseou-se na biblioteca gráfica da Microchip, e em particular nas instruções geradas no interior das funções *GOLDrawCallback* e *GOLMsgCallback*.

De seguida são apresentados esquematicamente na Figura 4.22 os estados e ecrãs associados desenvolvidos já adaptados à *Explorer 16*.

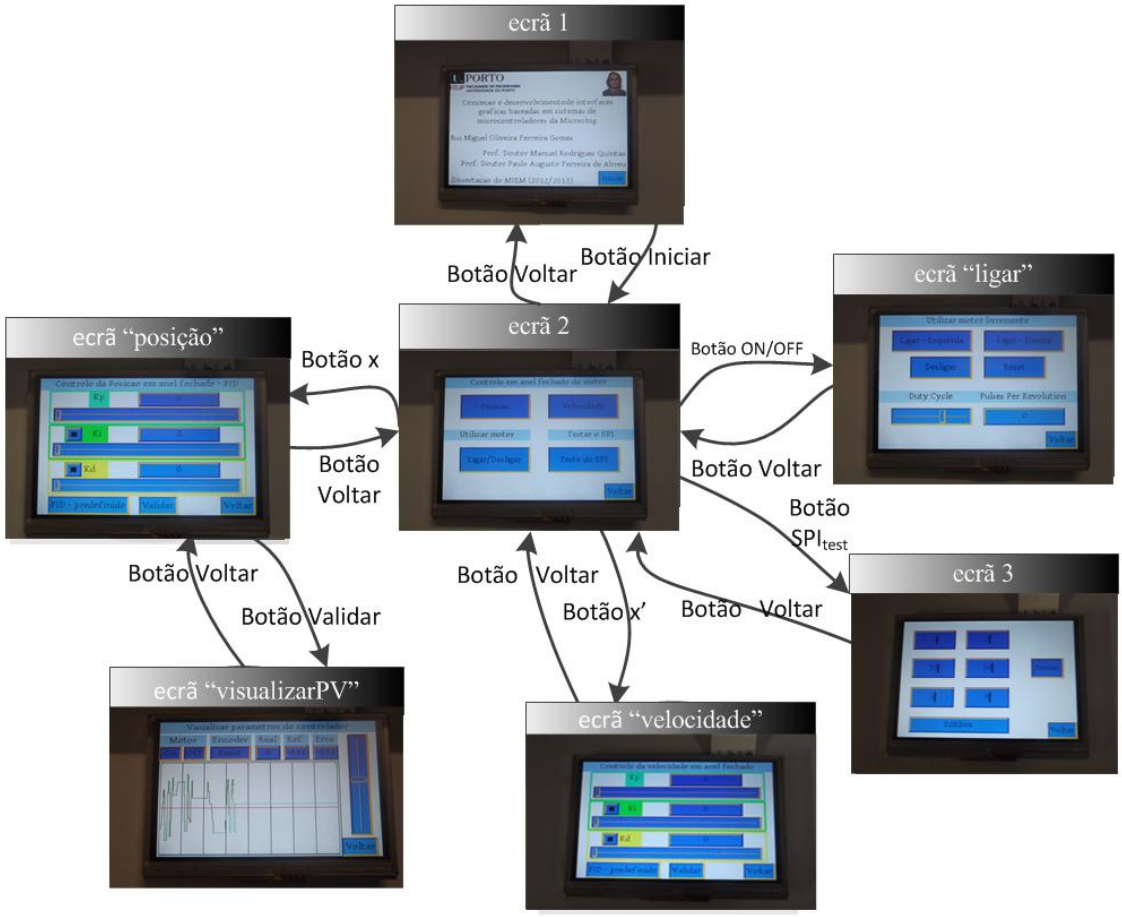


Figura 4.22 – Organização de estados da interface gráfica

As denominações dos ecrãs na Figura 4.22 correspondem às designações introduzidas na programação da interface gráfica (Figura 4.23).

```
void (*CreateFunctionArray[NUM_GDD_SCREENS]) (void) =

{
&Createecra1,
&Createecra2,
&Createecra3,
&Createposicao,
&Createvelocidade,
&Createligar,
&CreatevisualizarPV,
};
```

Figura 4.23 – Designação dos ecrãs desenvolvidos.

A organização da interface inicia-se com o "ecra1" de apresentação, acessível no arranque do sistema. No anexo B é possível visualizar o código associado.

Comuta para o "ecra2", ao tocar-se no botão "Iniciar", que constitui a raiz da interface gráfica, isto é, é acessível a partir de qualquer menu diretamente com exceção do ecrã "visualizarPV" que no código desenvolvido constitui um submenu do ecrã "posicao". O "ecra2" inclui um conjunto de funções acessíveis conforme a Tabela 4.1.

Tabela 4.1 – Funções acessíveis a partir do "ecra2"

Botão do "ecra2"	
"Ligar/Desligar"	"ligar"
"Posição"	"posicao"
"Velocidade"	"velocidade"
"Teste de SPI"	"visualizarPV"

O ecrã "ligar" (Figura 4.24) contém as funções de comando do motor em anel aberto.

Figura 4.24 – Organização de estados da interface gráfica: ecrã "ligar"

Os ecrãs "posicao" e "velocidade" (Figura 4.25 **a** e **b**) destinam-se, respetivamente, ao ajuste dos parâmetros do controlador de posição e do de velocidade.

(a) (b)

Figura 4.25 – Organização de estados da interface gráfica: (a) ecrã "posicao" e (b) ecrã "velocidade"

O código associado à definição do objeto GOL *slider* correspondente ao ajuste da constante proporcional do controlador é apresentado na Figura 4.26.

```
SLIDER *pSLD_36;
  pSLD_36 = SldCreate(  SLD_36,                          //Nome
                20,                                      //Esquerda
                53,                                      //Topo
                299,                                     //Direita
                78,                                      //Base
                SLD_DRAW | SLD_DRAW_THUMB,         //Estado
                1000,                                    //Gama
                1,                                       //Resolução
                10,                                      //Posição inicial
               defscheme                                 //scheme
             );
```

Figura 4.26 – Código de programação para definição *widget* tipo *slider* do ecrã "posicao"

O estilo do objeto utilizado foi o esquema por defeito, no entanto pode ser alterado e assim implementadas outras cores e/ou fontes através da API *GolCreateScheme* (subcapítulo 4.1).

No ecrã "velocidade" não foi implementado o código de controlo associado. Ocorre a mesma situação para o controlo integral e derivativo de posição angular. De referir que há um ecrã para teste comunicação SPI descrito mais adiante.

Os últimos menus referidos são acessíveis a partir do "ecra2" e dispõem de tecla para voltar a este mesmo menu.

O ecrã "visualizarPV" (Figura 4.27) que contém *slider* associado ao controlo de posição angular do veio do motor e *strip chart* para monitorização, é implementado a partir da tecla "Validar" do ecrã "posicao" e pode regressar-se a este menu tocando na tecla "Voltar".

Figura 4.27 – Organização de estados da interface gráfica: ecrã “visualizarPV”

O *strip chart* referido, cujo excerto do código mais significativo se encontra no anexo C, não é desenvolvido recorrendo à função automaticamente gerada pelo GDDx no *software* MPLABx: o GDDx_*Event_Handler*.c, mas inserindo diretamente o código na função *main*, efetuando assim a programação direta com a biblioteca gráfica.

Os vários *widgets* dentro de cada menu são implementados na função *GOLDrawCallback* e chamados na função *GOLMsgCallback* por um comando *if* (Figura 4.28).

```
if( (pObj-> ID ==(SLD_1)) && (objMsg == SLD_MSG_INC) )
```

Figura 4.28 – Comando if para tipo e estado do objeto na função *GOLMsgCallback*

Este comando verifica simultaneamente o tipo de objeto e o seu estado, executando de seguida o código que se pretende implementar cuja extensão em alguns casos é significativa.

O “ecra3” representado esquematicamente na Figura 4.29 inclui o teste de comunicação SPI.

Figura 4.29 – Organização de estados da interface gráfica: “ecra3”

Se o utilizador usar a tecla “Voltar” entra no “ecra2”.

De referir que, no caso da mudança entre ecrãs o código gerado no GDDx é simples e corresponde a: “*GDDDemoGoToScreen(1);*”, em que o dígito corresponde ao número do ecrã.

4.4 Resultados

O ecrã desenvolvido no ambiente de *hardware* com microcontrolador de 16 *bits* para a função de menu inicial é o representado na Figura 4.30.

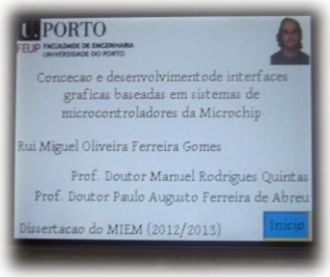


Figura 4.30 – Ecrã inicial

Este menu contém indicações sobre a designação do projeto e instituição de tutela, ano de realização, nomes do mestrando e orientadores de dissertação. O logótipo da instituição e a imagem do mestrando correspondem a imagens do tipo *Bitmap* cuja inserção através da ferramenta de *software* GDDx da Microchip é bastante expedita.

Quando o utilizador pressiona em “Inicio” (no vértice inferior direito do ecrã táctil) ocorre a mudança para o “Home” da aplicação.

De referir que o GDDx possibilita a inserção de imagens com a dimensão do LCD e a ativação de mudança de menu por toque em qualquer posição do ecrã, não estando o programador condicionado à inserção de botões para esse efeito. Esta função é bastante positiva em imagens introdutórias pois permite poupança de memória, tendo sido implementada com sucesso somente no PIC32.

Após a apresentação do menu inicial é descrito o menu *home* representado na Figura 4.31.

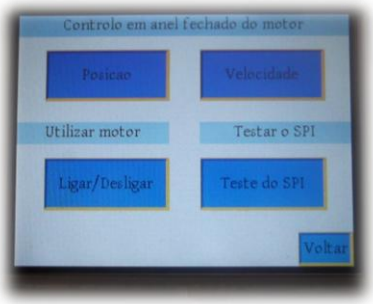


Figura 4.31 – Ecrã *home*

O utilizador é colocado face a um conjunto de opções de comando e monitorização do motor elétrico, que serão abordadas individualmente, a seguir à apresentação do ecrã *home*.

O ecrã é constituído por quatro botões que permitem aceder às janelas de controlo em anel fechado de "Posição" e "Velocidade", e à janela de utilização do motor em anel aberto. Conforme referido, a quarta função acessível é de teste do SPI e é implementável via *software*. No vértice inferior direito está situado o botão "Voltar".

A janela de utilização em anel aberto do acionamento (Figura 4.32) é composta por sete objetos GOL dos tipos *button*, *slider*, *editbox*, *window* e *static text*.

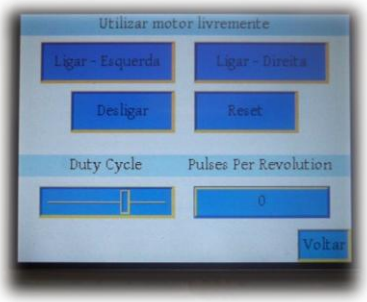


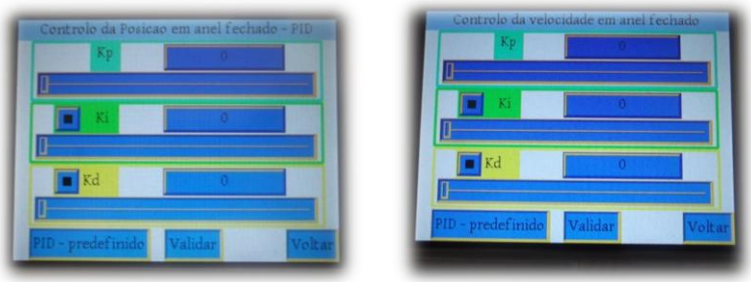
Figura 4.32 – Ecrã controlo da velocidade angular

A barra no topo do ecrã indica a função geral disponível, que corresponde à possibilidade do operador ligar e desligar o atuador, alterar o sentido de rotação do mesmo e implementar a variação do *duty cycle* do PWM associado ao controlo do motor CC, através do cursor do *slider*.

O botão “Ligar - Esquerda” e “Ligar - Direita” desempenham funções análogas com exceção do sentido de rotação que é ativado. Quando se pressiona “Ligar - Esquerda” o motor inicia o seu funcionamento com rotação no sentido direto. No caso de a opção ser “Ligar - Direita” o sentido de rotação é retrógrado. Para terminar a utilização do motor recorre-se ao botão “Desligar”.

É possível efetuar o “Reset” do *encoder* e conforme já referido alterar o *duty cycle* do PWM através do *slider*. No que concerne à monitorização, existe uma *editbox* que vai informando o utilizador em modo numérico do valor do contador associado ao QEI (*encoder*). Este menu permite regressar ao menu *home* através da tecla “Voltar”.

Foram desenvolvidos dois menus para ajuste dos parâmetros do controlador do motor (Figura 4.33 **a** e **b**).



(a) **(b)**

Figura 4.33 – Ecrã ajuste parâmetros controlador PID: (a) Posição e (b) Velocidade

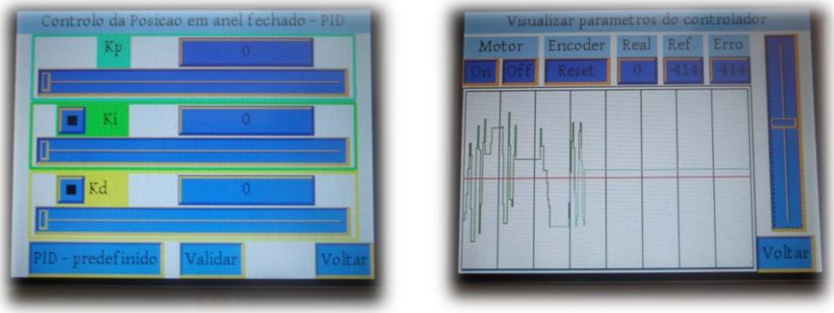
São análogos na estrutura e disposição dos objetos GOL, sendo diversos no propósito que num dos casos é ajustar o controlador PID para posição e no outro para a velocidade do veio do motor. Os ecrãs dispõem de todas as valências gráficas operacionais para ajuste do controlador PID de posição e velocidade. Foi implementado, ao nível da interface de comando (com o microcontrolador de 8 *bits*: PIC18LF4431), a função de controlo proporcional da posição em anel fechado.

Os menus “Controlo de Posição em anel fechado - PID” e “Controlo de Velocidade em anel fechado - PID” são constituídos por uma *window* que contém a designação da janela, três *sliders* do tipo horizontal, várias *static text* para inserção de texto, duas *check box* com seleção independente para as constantes da

componentes integral e derivativa do controlador PID e *editbox* para visualização numérica dos valores das constantes proporcional, integral e derivativa.

Inclui ainda a componente gráfica de um valor predefinido dos parâmetros do PID e um botão "Validar" para entrar no submenu de controlo e monitorização

O controlo proporcional de posição angular é implementado a partir do menu *home*, pressionando a tecla "Posição" que nos envia para o ecrã do controlador representado na Figura 4.34 **a**. O utilizador deve selecionar o valor da constante proporcional e de seguida selecionar "Validar" que ativa um outro menu que corresponde ao ecrã "visualizarPV" do diagrama de estados (Figura 4.34 **b**).



(a) (b)

Figura 4.34 – Ecrãs de ajuste e implementação do controlo proporcional de posição motor

Neste outro ecrã é possível introduzir um valor de referência para posição e monitorizar este valor e o resultado da ação de controlo sobre esta grandeza mecânica através de *strip chart* (Figura 4.35). É também possível visualizar numericamente os valores de referência, de erro e real da posição angular.

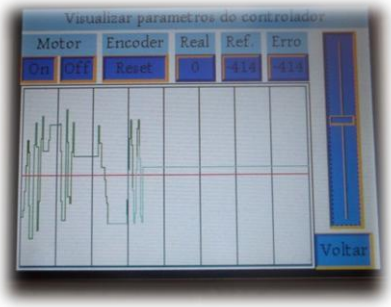


Figura 4.35 – Ecrã controlo proporcional de posição motor

Esta janela é constituída por um gráfico de monitorização (*strip chart*) e um conjunto de funções como ligar e desligar o motor e *reset* do *encoder*. O *slider* é do tipo vertical e na sua base encontra-se o botão "Voltar" ao menu "Controlo de Posição em anel fechado – PID".

Conforme referido na descrição das funções implementáveis a partir do *home*, este ecrã permite testar a comunicação SPI entre os microcontroladores de cada um dos módulos do protótipo (Figura 4.36).

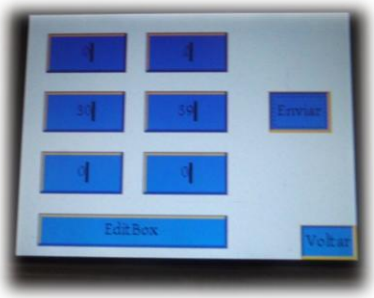


Figura 4.36 – Ecrã de teste SPI

Esta opção foi desenvolvida aquando da passagem da MEB para *Explorer* 16 a fim de testar a real disponibilidade da comunicação SPI nos dois sentidos e entendeu-se deixar esta função ativa, funcionando como *debuger*.

Graficamente é constituído por sete *editbox* e dois botões com texto. Nas *edit-box* é implementada a função EB_CARET que ativa um cursor no *widget*.

4.5 Conclusão

Este capítulo conclui a apresentação da biblioteca gráfica com as APIs base do desenvolvimento das interfaces gráficas.

O comando da posição e velocidade de operação do motor é implementável a partir de um conjunto de menus no ecrã táctil, sendo o inicial do tipo informativo; o segundo menu é do tipo *home*, a partir do qual se pode aceder a qualquer um dos outros ecrãs; um destes outros ecrãs permite ligar e desligar o motor para funcionar com sentido de rotação direto ou retrógrado a uma velocidade que o utilizador pode comandar através de um *slider* e monitorizar numericamente o

resultado. Um outro menu é para introdução dos valores das constantes k_P, k_I e k_D para controlo de posição em anel fechado do controlador PID.

Há um ecrã desenvolvido, acessível através do ecrã anteriormente descrito, para o controlo proporcional de posição do atuador e a monitorização através de *strip chart*, e um outro, utilizado para teste de comunicação SPI.

5 Conclusões e trabalhos futuros

O trabalho efetuado permitiu o desenvolvimento, baseado em dois microcontroladores de baixo custo, de uma interface HMI *touch* para o controlo *on*/*off*, do sentido de rotação, da posição e velocidade angulares de um motor CC com escovas e a monitorização de sinais relevantes.

Foi implementado com sucesso um protótipo constituído por duas partes modulares que incluem no módulo HMI, um microcontrolador de 16 *bits* e microprocessamento gráfico externo, e no outro módulo um microcontrolador de 8 *bits* para controlo por PWM do motor CC. Estes módulos comunicam por SPI e foram utilizados os periféricos de comunicação respetivos nos microcontroladores referidos.

Foi também implementada num microcontrolador de 32 *bits* a comunicação SPI num sentido (*out*), no modo 8 *bits* e o desenvolvimento de praticamente toda a componente gráfica da HMI final.

O uso de *touch screens* controlados a partir de microcontroladores constitui uma solução poderosa e económica em alternativa a HMI já disponíveis no mercado.

A implementação de uma solução deste tipo requer um conhecimento profundo a nível do *hardware* utilizado e da programação que é necessária efetuar, apesar dos fabricantes já disponibilizarem bibliotecas gráficas que constituem uma importante mais-valia na implementação.

O protótipo desenvolvido inclui ecrãs desenvolvidos não só com a aplicação GDDx, que utiliza a maioria das valências da biblioteca gráfica, mas também programando diretamente o recurso MPLABx com a biblioteca referida. A HMI possibilita o controlo do motor elétrico em anel aberto, o controlo proporcional da posição angular e a sua monitorização em tempo real através de *strip chart*.

Este trabalho de dissertação também pretende incentivar controlo de atuadores, a partir de objetos virtuais: *slider*, *dial* ou *button*, e monitorização em ecrã gráfico a cores e táctil. A unidade de processamento gráfico pode ser encarada como um periférico do microcontrolador e deve ser utilizada tal como outros periféricos, na implementação de sistemas de e em Engenharia Mecânica.

Trabalhos futuros

Neste protótipo é possível integrar mais potencialidades, designadamente o desenvolvimento e integração de interfaces para sensores analógicos e/ou digitais.

A placa *Explorer* 16 inclui um sensor de temperatura e um LCD de carateres que podem ser configurados respetivamente como módulo sensorial e de verificação de dados monitorizados.

O *hardware* do módulo HMI pode ser alterado e implementada uma arquitetura para interface gráfica com PIC32, obtendo-se uma solução que não inclua placas de desenvolvimento.

É possível a produção de placas de circuito impresso para implementação do sistema.

O algoritmo de controlo do motor pode ser implementado com componente integral e derivativa, permitindo o controlo por PID da posição e da velocidade angulares do atuador em anel fechado.

O protótipo pode ser associado a um maquinismo: para transformação do movimento de rotação do motor em translação e dotado de uma parte modular sensorial para deteção de posição, que permita a monitorização e o posicionamento em cinco posições distintas de um carro num eixo linear.

6 Referências

[1] http://arstechnica.com/gadgets/2013/04/from-touch-displays-to-the-surface-a-brief-history-of-touchscreen-technology/

[2] http://www.ti.com

[3] Harris, D., Harris S., "*Digital Design and Computer Architecture*" 2nd *Edition*, *Morgan Kaufmann*, 2013.

[4] http://isawwsymposium.com/wp-content/uploads/2012/07/WWAC2012-invited_BillHollified_HighPerformanceHMIs_paper.pdf

[5] *Implementing a Cost-Effective Human-Machine Interface for Home Appliances*. 2009; Disponível em: http://www.altera.com/literature/wp/wp-01083-cost-effective-hmi-home-appliances.pdf

[6] http://www.analog.com/library/analogdialogue/archives/35-04/touchscreen/

[7] http://www.digikey.com/Web%20Export/Supplier%20Content/NewhavenDisplay_757/pdf/newhaven-touchpanel-guide.pdf?redirected=1

[8] Microchip DS01368A *datasheet*, 2011.

[9] Microchip DS39969B *datasheet*, 2010.

[10] http://www.microcontrollerboard.com/pic_serial_communication.html#SerandParComm

[11] Sousa, D. R., Souza, D. J., "DESBRAVANDO O PIC24 Conheça os Microcontroladores de 16 *bits*", Editora Érica, 2008.

[12] Microchip, PIC32 *family datasheet*, p. 337.

[13] Microchip, DS39905B *datasheet*,. p. 7.

[14] http://www.anglia-displays.com/displays/tft/datasheets/MTF-TQ35SP811-AV.pdf

[15] Azevedo, C., "Comando e Monitorização de Sistemas de Actuação Via Redes Wireless – ZigBee", FEUP, 2010.

[16] http://www.microchip.com

[17] Pereira, F., "Microcontroladores PIC Programação em C", 7ª Edição, Editora Érica, 2009.

[18] Microchip, MeB *datasheet*, 2010.

[19] Di Jasio, L., "*Programming 32-bit Microcontrollers in C Exploring the PIC32*", *Newnes*, 2008.

[20] http://dei-s1.dei.uminho.pt/labsim/SimLab/controlo/PID_D.html

[21] Ogata, K., "*Discrete-Time control systems*", *Prentice Hall Int*., 1987. p. 203

(Todos os *websites* foram verificados a 2013-08-29)

Data sheet

- Microchip AN1182
- Microchip AN1136
- Microchip AN1227
- Microchip AN1246
- Microchip AN1368
- Microchip AN964
- Microchip AN696
- Microchip AN718
- Microchip AN905
- Microchip *Graphics Library Version* 3.06.02
- Microchip GDD X *User Guide*
- SOLOMON SYSTECH SSD1926
- Microtips *Technology* MTF-TQ35SP811-AV
- Microchip PIC32MX5XX/6XX/7XX
- Microchip 32-*BIT LANGUAGE TOOLS LIBRARIES*
- Microchip PIC32 ETHERNET STARTER KIT USER'S GUIDE
- Microchip MULTIMEDIA EXPANSION BOARD USER'S GUIDE
- Microchip PIC24FJ256GA110 *Family*
- Microchip EXPLORER 16
- Microchip PIC18F2331/2431/4331/4431
- RENESAS HD74LS00
- *STMicroelectronics* L298

ANEXOS

ANEXO A:

/**Código exemplo comentado com dois botões e *slider* para a função *GOLMsg-Callback*()**/

/**INÍCIO**/

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg){
WORD objectID;
SLIDER *pSldObj;

objectID = GetObjID(pObj);
                // coloca ID do objeto ativo
                // verifica se a mensagem/comando do utilizador é para o primeiro botão
if (objectID == ID_BTN1) {
                // código para o 1.º botão
                // verifica se o botão é pressionado
                // objMsg corresponde à mensagem/comando do utilizador a partir objeto
        if (objMsg == BTN_MSG_PRESSED) {
                // se botão é pressionado o slider decrementa uma posição
                // o apontador slider pointer vai conter ID_SLD1
        pSldObj = (SLIDER*)GOLFindObject(ID_SLD1);
                // a posição do slider vai diminuir um valor correspondente à resolução definida
        SldDecPos(pSldObj);
                // define o estado para redesenhar o slider thumb para nova posição
        SetState(pSldObj, SLD_DRAW_THUMB);
        }
        }
                // verifica se a mensagem é para o 2.º botão
if (objectID == ID_BTN2) {
```

```
            // código para o 2.º botão
            // verifica se o estado do botão é pressionado
        if (objMsg == BTN_MSG_PRESSED) {
            // se o botão está pressionado o valor do slider incrementa
        pSldObj = (SLIDER*)GOLFindObject(ID_SLD1);
            // a posição do slider vai diminuir um valor correspondente à resolução definida
        SldIncPos(pSldObj);

        SetState(pSldObj, SLD_DRAW_THUMB);
        }
        }
            // deve devolver 1 para atualizar estado dos botões
return 1;

/**FIM**/
```

/**Código exemplo comentado com barras indicadoras da variação *slider* para *GOLDrawCallback*()**/

/**INÍCIO**/

```
WORD GOLDrawCallback(){
 WORD value, y, x;                          // Variáveis para definir posição slider
 static WORD prevValue = 0;                 // Retem o valor anterior do slider

 if (update)
{
                value = SldGetPos (pSLD_1);
                SetColor(BLACK);
                if (value < prevValue)
                {
                  while (prevValue > value)
                   {
                         y= (prevValue*prevValue)/110;
                         x=(prevValue*2);          //Barras a cada 6 pixel com largura de 4
                         x=x-(x%6);
                         Bar(x+60,100-y,x+64,100);
                         prevValue -=3;
                        }
                }
                else
                {   while (prevValue < value)
                   {
                          if ( prevValue < 60)
                          {SetColor(BRIGHTGREEN);
                          }
                          else if ((  prevValue < 80) && (  prevValue >= 60))
                                        {SetColor(BRIGHTYELLOW);
                                        }
                                        else if ( prevValue >= 80)
                                        {{SetColor(BRIGHTRED);
                                        }
                           y = (prevValue*prevValue)/110;
                           x= (prevValue*2);
                           x=x-(x%6);
                           Bar(x+60,100-y,x+64,100);
                           prevValue +=3;
                        }
```

```
                }
                update = 0;
                }
//              return 1;
}
```

/**FIM**/

ANEXO B:

/**Código gerado pelo GDDx para o ecrã "Menu inicial"**/

/**INÍCIO**/

```
void Createecra1(void)
{
   GOLFree();
   SetColor(RGBConvert(248, 252, 248));
   ClearDevice();


    if(defscheme != NULL) free(defscheme);
      defscheme = GOLCreateScheme();

   defscheme->Color0 = RGBConvert(32, 168, 224);
   defscheme->Color1 = RGBConvert(16, 132, 168);
   defscheme->TextColor0 = RGBConvert(24, 24, 24);
   defscheme->TextColor1 = RGBConvert(248, 252, 248);
   defscheme->EmbossDkColor = RGBConvert(248, 204, 0);
   defscheme->EmbossLtColor = RGBConvert(24, 116, 184);
   defscheme->TextColorDisabled = RGBConvert(128, 128, 128);
   defscheme->ColorDisabled = RGBConvert(208, 224, 240);
   defscheme->CommonBkColor = RGBConvert(208, 236, 240);
   defscheme->pFont = (void*)&Gentium_16;

   if(esquemaInicio != NULL) free(esquemaInicio);
      esquemaInicio = GOLCreateScheme();

   esquemaInicio->Color0 = RGBConvert(248, 252, 248);
   esquemaInicio->Color1 = RGBConvert(248, 252, 248);
   esquemaInicio->TextColor0 = RGBConvert(24, 24, 24);
   esquemaInicio->TextColor1 = RGBConvert(248, 252, 248);
   esquemaInicio->EmbossDkColor = RGBConvert(248, 204, 0);
```

```
esquemaInicio->EmbossLtColor = RGBConvert(248, 252, 248);
esquemaInicio->TextColorDisabled = RGBConvert(248, 252, 248);
esquemaInicio->ColorDisabled = RGBConvert(248, 252, 248);
esquemaInicio->CommonBkColor = RGBConvert(248, 252, 248);
esquemaInicio->pFont = (void*)&Gentium_16;


BUTTON *pBTN_22;
pBTN_22 = BtnCreate(  BTN_22, //name
              268, //left
              208, //top
              319, //right
              239, //bottom
              0, //radius
              BTN_DRAW, //state
              NULL, //bitmap
              (XCHAR*)ecra1_BTN_22text, //text
             defscheme //scheme
            );

if(pBTN_22==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

PICTURE *pPCB_87;
pPCB_87 = PictCreate(  PCB_87, //name
              0, //left
              0, //top
              130, //right
              43, //bottom
              PICT_DRAW, //state
              1, //scale
              (void*)&UP_logo10, //bitmap
             defscheme //scheme
            );

if(pPCB_87==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
```

```
}

STATICTEXT *pSTE_91;
pSTE_91 = StCreate(  STE_91, //name
              0, //left
              131, //top
              319, //right
              153, //bottom
              ST_DRAW, //state
              (XCHAR*)ecra1_STE_91text, //text
             esquemaInicio //scheme
            );

if(pSTE_91==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

STATICTEXT *pSTE_92;
pSTE_92 = StCreate(  STE_92, //name
              0, //left
              163, //top
              319, //right
              185, //bottom
              ST_DRAW | ST_RIGHT_ALIGN, //state
              (XCHAR*)ecra1_STE_92text, //text
             esquemaInicio //scheme
            );

if(pSTE_92==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

STATICTEXT *pSTE_93;
pSTE_93 = StCreate(  STE_93, //name
              0, //left
              185, //top
              319, //right
              207, //bottom
```

```
                ST_DRAW | ST_RIGHT_ALIGN, //state
                (XCHAR*)ecra1_STE_93text, //text
                esquemaInicio //scheme
              );

  if(pSTE_93==NULL)
  {
    CreateError(0);
    while(1); //Fatal Error, Check for memory leak or heap size
  }

  PICTURE *pPCB_86;
  pPCB_86 = PictCreate(  PCB_86, //name
                277, //left
                0, //top
                319, //right
                53, //bottom
                PICT_DRAW, //state
                1, //scale
                (void*)&Rui_foto10, //bitmap
                defscheme //scheme
              );

  if(pPCB_86==NULL)
  {
    CreateError(0);
    while(1); //Fatal Error, Check for memory leak or heap size
  }

  STATICTEXT *pSTE_95;
  pSTE_95 = StCreate(  STE_95, //name
                0, //left
                56, //top
                319, //right
                78, //bottom
                ST_DRAW | ST_CENTER_ALIGN, //state
                (XCHAR*)ecra1_STE_95text, //text
                esquemaInicio //scheme
              );

  if(pSTE_95==NULL)
  {
```

```
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

STATICTEXT *pSTE_96;
pSTE_96 = StCreate(  STE_96, //name
                0, //left
                75, //top
                319, //right
                97, //bottom
                ST_DRAW | ST_CENTER_ALIGN, //state
                (XCHAR*)ecra1_STE_96text, //text
               esquemaInicio //scheme
              );

if(pSTE_96==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

STATICTEXT *pSTE_97;
pSTE_97 = StCreate(  STE_97, //name
                0, //left
                95, //top
                319, //right
                117, //bottom
                ST_DRAW | ST_CENTER_ALIGN, //state
                (XCHAR*)ecra1_STE_97text, //text
               esquemaInicio //scheme
              );

if(pSTE_97==NULL)
{
  CreateError(0);
  while(1); //Fatal Error, Check for memory leak or heap size
}

STATICTEXT *pSTE_98;
pSTE_98 = StCreate(  STE_98, //name
                0, //left
                220, //top
```

```
                    230, //right
                    239, //bottom
                    ST_DRAW | ST_CENTER_ALIGN, //state
                    (XCHAR*)ecra1_STE_98text, //text
                   esquemaInicio //scheme
                  );

   if(pSTE_98==NULL)
   {
     CreateError(0);
     while(1); //Fatal Error, Check for memory leak or heap size
   }


}
```

/**FIM**/

ANEXO C:

/**Código implementado no MPLABx XC16 para o desenho do gráfico monitorização (API *GOLDrawCallback*)**/

/**INÍCIO**/

```
void PRealGrafico(void)
{
   SHORT         x, y;
   SHORT         sy, ey;
   SHORT         *ptr;
   SHORT         counter;
   static SHORT    pos;

   // Remover gráfico
   SetColor(WHITE);
   ptr = PRealBuffer + pos;
   sy = *ptr++;
   for(x = GR_RIGHT; x >= GR_LEFT; x--){
      if(ptr == (PRealBuffer + PReal_BUFFER_SIZE))
         ptr = PRealBuffer;
      ey = *ptr++;
      if(ey > sy){
         for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
            PutPixel(x, y);
      }
      else{
         for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
            PutPixel(x, y);
      }
      sy = ey;
   }

   /*Desenho da grelha do gráfico*/
   // Desenha quadrado exterior em azul cyan
```

```
   SetColor(CYAN);
   (Line(GR_LEFT, GR_TOP, GR_LEFT, GR_BOTTOM));
   (Line(GR_LEFT, GR_BOTTOM, GR_RIGHT, GR_BOTTOM));
   (Line(GR_RIGHT, GR_TOP, GR_RIGHT, GR_BOTTOM));
   (Line(GR_RIGHT, GR_TOP, GR_LEFT, GR_TOP));


   // Desenho das semirretas verticais
   for(x = GR_LEFT + ((GR_RIGHT - GR_LEFT) >> 3); x < GR_RIGHT - 10; x += (GR_RIGHT -
GR_LEFT) >> 3)  // x < GR_RIGHT – 10, o - 10 foi colocado para as divisoes nao conduzirem a
//um erro de 1 pixel
   WAIT_UNTIL_FINISH(Bar(x, GR_TOP, x, GR_BOTTOM));


   // Desenho das semirretas horizontais
   for(y = (GR_TOP) + ((GR_BOTTOM - (GR_TOP)) >> 1); y < GR_BOTTOM; y += (GR_BOTTOM -
(GR_TOP)) >> 1)
   WAIT_UNTIL_FINISH(Bar(GR_LEFT, y, GR_RIGHT, y));




   pos -= PGraph_MOVE_DELTA;
   if(pos < 0)
   pos = PReal_BUFFER_SIZE - 1;


   // Copiar novos dados para o buffer temporário
   ptr = PRealBuffer + pos;
   for(counter = PGraph_MOVE_DELTA - 1; counter >= 0; counter--){
      *ptr++ = PTempRealBuffer[counter];
      if(ptr == (PRealBuffer + PReal_BUFFER_SIZE))
        ptr = PRealBuffer;
   }


   // Desenhar gráfico
   SetColor(BRIGHTRED);
   ptr = PRealBuffer + pos;
   sy = *ptr++;
   for(x = GR_RIGHT; x >= GR_LEFT; x--){
      if(ptr == (PRealBuffer + PReal_BUFFER_SIZE))
         ptr = PRealBuffer;
      ey = *ptr++;
      if(ey > sy){
         for(y = sy + GR_TOP; y < ey + GR_TOP + 1; y++)
            PutPixel(x, y);
      }
      else{
```

```
            for(y = ey + GR_TOP; y < sy + GR_TOP + 1; y++)
                PutPixel(x, y);
        }
        sy = ey;
    }
}
```

/**FIM**/